Produktion und Logistik

RESEARCH

Matthias Walter

Multi-Project Management with a Multi-Skilled Workforce

A Quantitative Approach Aiming at Small Project Teams

Springer Gabler

Produktion und Logistik

Diese Reihe dient der Veröffentlichung neuer Forschungsergebnisse auf den Gebieten der Produktion und Logistik. Aufgenommen werden vor allem herausragende quantitativ orientierte Dissertationen und Habilitationsschriften. Die Publikationen vermitteln innovative Beiträge zur Lösung praktischer Anwendungsprobleme der Produktion und Logistik unter Einsatz quantitativer Methoden und moderner Informationstechnologie.

Kontakt
Professor Dr. Thomas S. Spengler
Technische Universität Braunschweig
Institut für Automobilwirtschaft
und Industrielle Produktion
Katharinenstraße 3
38106 Braunschweig

Matthias Walter

Multi-Project Management with a Multi-Skilled Workforce

A Quantitative Approach Aiming at Small Project Teams

Foreword by Prof. Dr. Jürgen Zimmermann

Springer Gabler

Matthias Walter
Clausthal-Zellerfeld, Germany

Dissertation Clausthal University of Technology, 2014
DE-104

Produktion und Logistik
ISBN 978-3-658-08035-8 ISBN 978-3-658-08036-5 (eBook)
DOI 10.1007/978-3-658-08036-5

Library of Congress Control Number: 2014954235

Springer Gabler

Printed on acid-free paper

Springer Gabler is a brand of Springer Fachmedien Wiesbaden
Springer Fachmedien Wiesbaden is part of Springer Science+Business Media
(www.springer.com)

Foreword

Due to the ongoing global change from sellers' to buyers' markets, the share of industrial production and services that is provided in the form of project deliverables continues to rise. Estimates suggest that current annual budgets for projects amount to U.S.$12 trillion or 20 % of the world's gross domestic product. Hence, successful project management is of salient importance for an economy as a whole and for each single firm involved in projects.

From the perspective of a firm, the selection of projects and the staffing of the chosen projects are crucial for its success. In regard to project execution, a firm's prosperity relies on an efficient use of its employees and their skills, on well-performing project teams, on smooth project execution, and on content employees. When employees have to work in large teams, however, their individual performance may decline due to coordination problems and motivation losses.

So far, only a few optimization models and methods exist that offer decision support for project selection and staffing and that consider special characteristics of human resources. The work at hand, which mainly addresses the composition of an optimal project portfolio and the assignment of workers to small project teams, thus tackles challenging problems, whose resolution harbors great benefits. For the first time, this work devises a hierarchically integrated approach that supports decision makers who face selection and staffing problems given a multi-skilled workforce with heterogeneous skill levels. The approach, which was developed and also coded and thoroughly tested by Matthias Walter, comprises three stages that are associated with the following three questions:

(1) Which is the optimal project portfolio subject to a given workforce?

(2) How should the available workers be assigned to the selected projects?

(3) How can the workload of the employees be leveled assuming a matrix organization where work requirements originate from projects and from departments?

At the first stage, portfolio selection is considered. Here, a firm has to choose a set of projects that either have been requested by customers or have been proposed within the firm. The selection is constrained by the capacity of non-consumable resources hold by the firm. In his thesis, Matthias Walter outlines a model that explicitly accounts for the capacities of human resources. Furthermore, he examines different skill configurations, i.e., different cross-training strategies for workers. A new skill configuration is proposed that enlarges the flexibility of a workforce. In addition, a refined flexibility measure is devised which shows that the novel skill configuration is superior to established chaining configurations.

The focus of the thesis is on the second stage where the staffing problem is dealt with. In this staffing problem, a team of workers must be composed for each project of the firm. Each worker masters some skills and each project requires a subset of all mastered

skills. The objective is to minimize average project team size and, hence, to minimize the average number of project assignments per worker. The objective prevents that project teams suffer unduly from social loafing and that workers loose their focus. Since team size has not been addressed before, the problem formulation and the proposed solution methods make a valuable contribution to the field of multi-project management.

A postprocessing step at the last stage ensures that differences in working times among workers, which arise from the project assignments, are mitigated by allocating departmental workload. For this leveling task, a linear program is formulated and solved by a dedicated polynomial-time algorithm.

The comprehensive performance analysis of the thesis demonstrates that the three-stage approach is well suited even for large-scale planning problems. For instances that feature a large number of projects and workers, good solutions are determined in acceptable time. Consequently, the approach offers a great potential for practitioners.

Altogether, the work at hand is a convincing application of operations research methods to the vital field of project management and provides a notable advancement for the task of forming project teams.

Clausthal-Zellerfeld, June 2014 Prof. Dr. Jürgen Zimmermann

Preface

I wrote this thesis during my time at the Institute of Management and Economics at Clausthal University of Technology in Clausthal-Zellerfeld. In this time between 2008 and 2014, it was my aim to devise new ways that support decision makers who face challenges in multi-project management and that also support the members of the affected project teams. I tried to build these ways on a solid ground provided by operations research methods. My hope is that both decisions makers and project team members can benefit from the ideas and methods that I have developed.

Completing the thesis would not have been possible without the support of many people to whom I am very grateful. First of all, I would like to thank Prof. Dr. Jürgen Zimmermann, who sparked my interest in operations research when I attended his lectures during my undergraduate studies and who gave me the opportunity to extensively explore the fascinating field of modeling and mathematical optimization as a research assistant and Ph.D. student in his department. As the supervisor and chief reviewer of my thesis, he granted me complete freedom to bring forward own ideas, to follow and abandon them. He had always time to discuss these ideas and he provided valuable help, especially when it became mathematically tricky.

I am also very thankful to Prof. Dr. Christoph Schwindt, who kindly agreed to be the second reviewer. He supported my work many times by sharing his vast knowledge. Whether I asked him at 9 a.m. or 9 p.m., he always took the time to give helpful advice and to point to suitable literature.

The problems that I tackle in this thesis have a real background. This background was brought to my attention by Roger Mähr and Martin Rudolph, managing directors of Tiba Technologieberatung GmbH in Berlin. I would like to thank both of them for the pleasant and trustworthy cooperation, for their willingness to share their experience in project management gained in various industries and their knowledge of software for project management tasks.

I owe special thanks to Dr. Christian Heimerl and Prof. Dr. Rainer Kolisch from Technische Universität München who provided valuable insight in efficient data structures for the generalized network simplex method. My implementation of this method profited considerably from the insight.

For an enjoyable working atmosphere, mutual support, enriching discussions, and many joyful moments, I am very grateful to my colleagues within the Department for Operations Research at the Institute of Management and Economics: I would like to thank Sascha Effenberger, Carsten Ehrenberg, Alexander Franz, Dr. Thorsten Gather, Dr. Claas Hemig, Stefan Kreter, PD Dr. Julia Rieck, Marco Schulze, and Katrin Tormann. Support came also from student assistants within the department and from students whose seminars or final year projects dealt with topics that were related to my thesis.

It was a great pleasure to work at the Institute of Management and Economics. Hence, I thank all colleagues, especially former and current members of the “Mensa crew” for

relaxing and refreshing lunch breaks. Additionally, I would like to thank all those people and institutions of my university that provided services vital for my life as a researcher. The institutions include, in particular, the canteen, the library, and the computing center.

For proofreading and a critical review of a first draft and for many suggestions for improvements I am very much indebted to Carsten Ehrenberg, Alexander Franz, Stefan Kreter, and Dr. Thomas Walter.

Finally, I would like to thank my family, particularly my parents, my brother Thomas, and my uncle Manfred. Without their support, this work and many other activities during the last years would not have been possible.

Clausthal-Zellerfeld, June 2014 Matthias Walter

Contents

List of figures

List of tables

List of algorithms

List of symbols

Acronyms

DROP	Drop procedure
GNS	Generalized network simplex
GRAP	Greedy randomized assignment procedure
ISAP	Iterated simultaneous assignment procedure
LP	Linear program/linear programming
MIP	Mixed-integer linear program/mixed-integer linear programming
PIP	Pure integer linear program/pure integer linear programming
ROUND	Rounding procedure

Miscellaneous

$\emptyset$	Empty set
$:=$	Equal by definition
$\lfloor a \rfloor$	Largest integer less than or equal to a (floor function)
$\mathbb{N} = \{0, 1, 2, 3, \ldots\}$	Set of natural numbers
$\mathbb{R}$	Set of real numbers

Models and solution methods

$a_{kp} \in [0, 1]$	Real-valued decision variable that equals 1 if worker k is selected for project p, and 0 otherwise
b_p	Benefit of project p
$\mathcal{C}$	List of pairs (k, p) for which dropping is considered

D	Number of departments of a firm
$\mathcal{D} = \{1, \ldots, D\}$	Set of departments of a firm
d	Department
$d(k)$	Department to which worker k belongs
$\Delta_{kk'} \in \mathbb{R}_{\geq 0}$	Decision variable that registers the amount of time that worker k works more than worker k'
$\Delta_{kk't} \in \mathbb{R}_{\geq 0}$	Decision variable that registers the amount of time that worker k works more than worker k' in period t
$f_{kt}^{\text{dep}} \in \mathbb{R}_{\geq 0}$	Decision variable that represents the flow of working time from worker k to his department in period t
$f_{kpt}^{\text{proj}} \in \mathbb{R}_{\geq 0}$	Decision variable that represents the flow of working time from worker k to project p in period t
K	Number of workers of a firm
$\mathcal{K} = \{1, \ldots, K\}$	Set of workers of a firm
$\mathcal{K}_d$	Set of workers that belong to department d
$\mathcal{K}_p^{\text{assigned}}$	Set of workers that are already assigned to the ongoing project p, $p \in \mathcal{P}^{\text{ongoing}}$ ($\mathcal{K}_p^{\text{assigned}} = \emptyset$ for $p \notin P^{\text{ongoing}}$)
$\mathcal{K}_p^{\text{frac}}$	Set of workers who are suitable for project p and whose corresponding variable x_{kp} has not been fixed to 0 or 1, subset of $\mathcal{K}_p^{\text{suit}}$
$\mathcal{K}_p^{\text{suit}}$	Set of workers that are suitable for project p
$\mathcal{K}_p^{\text{suit,rem}}$	Set of remaining workers that can contribute to project p
$\mathcal{K}_s$	Set of workers that master skill s
k	Worker
l_{ks}	Level with which worker k masters skill s
LB_p^{Drop}	Local lower bound on the number of workers that are needed to accomplish all requirements of project p, calculated in the course of the heuristic DROP
LB_{ps}^{Drop}	Local lower bound on the number of workers that are needed to accomplish all requirements of project p for skill s, calculated in the course of the heuristic DROP
LB_p^{glob}	Global lower bound on the number of workers that are needed to accomplish all requirements of project p

LB_{ps}^{glob}	Global lower bound on the number of workers that are needed to accomplish all requirements of project p for skill s
LB_{p}^{loc}	Local lower bound on the number of workers that are additionally needed to accomplish all requirements of project p
LB_{ps}^{loc}	Local lower bound on the number of workers that are additionally needed to accomplish all requirements of project p for skill s
$\text{LP}_{t}^{\text{rem}}$	Linear program that remains for period t if all binary variables x_{kp} in the workforce assignment problem are fixed
$\tilde{P}$	Number of projects that can be selected
$\tilde{\mathcal{P}} = \{1, \ldots, \tilde{P}\}$	Set of projects that can be selected
$\mathcal{P}^{\text{must}}$	Set of projects that must be selected
$\mathcal{P}^{\text{ongoing}}$	Set of ongoing projects, which must be continued
P	Number of projects that will be executed in the planning horizon
$\mathcal{P} = \{1, \ldots, P\}$	Set of projects that will be executed in the planning horizon
$\hat{\mathcal{P}}_{k}^{\text{suit}}$	Set of projects that are suitable for worker k, subset of $\mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}}$
$\mathcal{P}_{k}^{\text{suit}}$	Set of projects that are suitable for worker k, subset of $\mathcal{P}$
$\hat{\mathcal{P}}_{k}^{\text{suit}}(t)$	Set of projects that are suitable for worker k and executed in period t, subset of $\hat{\mathcal{P}}_{k}^{\text{suit}}$
$\mathcal{P}_{k}^{\text{suit}}(t)$	Set of projects that are suitable for worker k and executed in period t, subset of $\mathcal{P}_{k}^{\text{suit}}$
$\mathcal{P}^{\text{toRound}}$	Set of projects for which at least one corresponding variable x_{kp} is fractional and can be rounded up
$\mathcal{P}^{\text{toStaff}}$	List of projects that have not been completely staffed
p	Project
R_{kt}	Availability of worker k in period t
R_{kt}^{rem}	Remaining availability of worker k in period t
r_{pst}	Requirement of project p for skill s in period t
r_{pst}^{rem}	Remaining requirement of project p for skill s in period t
rd_{dt}	Requirement of department d in period t

rd_{dt}^{rem}	Remaining requirement of department d in period t
S	Number of skills
$\mathcal{S} = \{1, \ldots, S\}$	Set of skills
$\mathcal{S}_k$	Set of skills that are mastered by worker k
$\mathcal{S}_{kp}^{\text{match}}$	Set of matching skills between worker k and project p
$\mathcal{S}_{kp}^{\text{match,rem}}$	Set of remaining matching skills between worker k and project p
$\mathcal{S}_p$	Set of skills that are required by project p
$\mathcal{S}_p^{\text{rem}}$	Set of skills that are required by project p and whose workload has not been completey allocated to workers
s	Skill
$suit_{kp}$	Suitability of worker k for project p
T	Number of periods in the planning horizon of a firm
$\mathcal{T} = \{1, \ldots, T\}$	Set of periods, planning horizon of a firm
$\mathcal{T}_p$	Set of periods in which project p is executed
t	Period
t_p^{finish}	Finish period of project p
t_p^{start}	Start period of project p
$totR_{dt}^{\text{rem}}$	Total remaining availability of all workers that belong to department d in period t
us_{kp}	Unsuitability of worker k for project p
w_1, w_2, w_3	Objective function weights
$x_{kp} \in \{0, 1\}$	Decision variable that equals 1 if worker k is assigned to project p, and 0 otherwise
$\boldsymbol{x}$	Matrix that contains all decision variables x_{kp}
$\hat{y}_{kpst} \in \mathbb{R}_{\geq 0}$	Decision variable that represents the time worker k performs skill s for project p in period t, $p \in \hat{\mathcal{P}}_k^{\text{suit}}$
$\hat{\boldsymbol{y}}$	Matrix that contains all decision variables $\hat{y}_{kpst}$
$y_{kpst} \in \mathbb{R}_{\geq 0}$	Decision variable that represents the time worker k performs skill s for project p in period t, $p \in \mathcal{P}_k^{\text{suit}}$

$\boldsymbol{y}$	Matrix that contains all decision variables y_{kpst}
$yd_{kt} \in \mathbb{R}_{\geq 0}$	Decision variable that represents the time worker k accomplishes departmental workload in period t
$z_p \in \{0, 1\}$	Decision variable that equals 1 if project p is selected, and 0 otherwise
$\boldsymbol{z}$	Vector that contains all decision variables z_p

Generalized network simplex method

A	Set of arcs
a	Arc, also denoted by $\langle i, j \rangle$
a^{extra}	Extra arc of a 1-tree, also denoted by $\langle \alpha, \beta \rangle$
$\langle \alpha, \beta \rangle$	Extra arc of a 1-tree, also denoted by a^{extra}
ArcAndOrient(i)	Number of the arc that links node i and its predecessor node pred(i); the orientation of this arc is specified by the sign of ArcAndOrient(i) (positive sign: outbound, negative sign: inbound)
c_{ij}	Cost per unit flow on arc $\langle i, j \rangle$
c_{ij}^{red}	Reduced costs of arc $\langle i, j \rangle$
$change_{ij}$	Change in the flow on arc $\langle i, j \rangle$ induced by a unit flow change on the entering arc
δ	Maximum absolute change of the flow on the entering arc
depth(i)	Depth of node i in a rooted tree or 1-tree
final(i)	Final node of node i in a rooted tree or 1-tree
$\langle i, j \rangle$	Arc with from-node i and to-node j, also denoted by a
$\langle k, l \rangle$	Entering arc (arc that enters the basis)
μ_{ij}	Gain of arc $\langle i, j \rangle$
N	Set of nodes
onNewCycle(i) $\in \{0, 1\}$	Parameter that equals 1 if node i belongs to the new cycle of a bicycle induced by the entering arc, and 0 otherwise
onOldCycle(i) $\in \{0, 1\}$	Parameter that equals 1 if node i belongs to the cycle of a 1-tree, and 0 otherwise

$\text{onPath}(i) \in \{0, 1\}$	Parameter that equals 1 if node i belongs to the path between the two cycles of a bicycle, and 0 otherwise
π_i	Potential of node i (dual variable corresponding to node i)
$\langle p, q \rangle$	Leaving arc (arc that leaves the basis)
$\text{pred}(i)$	Predecessor of node i in a rooted tree or 1-tree
r_{tr}	Root node of the tree or 1-tree tr
$\text{revThread}(i)$	Reverse thread node of node i in a rooted tree or 1-tree
$\text{Root}(i)$	Root node of the tree or 1-tree to which node i belongs
$\text{SubtreeNodes}(i)$	Number of nodes in the subtree $tr(i)$
$\text{thread}(i)$	Thread node of node i in a rooted tree or 1-tree
tr	Tree or 1-tree
$tr(i)$	Subtree constituted by node i and its successors
u_{ij}	Upper bound on the flow on arc $\langle i, j \rangle$

Instance generation and numerical analysis

gap_μ	Average relative gap of solutions for the instances of a test set
obj_μ	Average objective function value of solutions for the instances of a test set
$opt.$	Number of instances from a test set for which proven optimal solutions were found
$\hat{P}$	Number of projects that are considered in the project selection problem $\left(\hat{P} = \left\|\mathcal{P}^{\text{ongoing}}\right\| + \left\|\mathcal{P}^{\text{must}}\right\| + \left\|\tilde{\mathcal{P}}\right\|\right)$
P_μ	Average number of selected projects in solutions for the instances of a test set
$pass^{\max}$	Maximum number of passes for a multi-start method
$pass_\mu$	Average number of completed passes for the instances of a test set
ρ^{dep}	Desired ratio of departmental workload to availability of department members in an instance
ρ^{proj}	Desired ratio of the total workload of all projects to workforce availability in an instance

$t^{\max}$	Time limit for a solution method
$time_{\mu}$	Average computation time for an instance of a test set
$util_{\mu}$	Average total utilization of workers in solutions for the instances of a test set

Chapter 1

Introduction

Multi-project management and human resource management are important pillars of business success. Multi-project management is crucial because the share of work that is accomplished within projects has increased steadily during the past 60 years and in some branches virtually all revenues are generated through projects (cf. Kerzner, 2013, pp. 47–63 and 25–27, respectively). As each project demands some resources, proper coordination of projects and resources is vital for a firm. With regard to resources, human resource management is a major factor because the demand for high-skilled workers has increased disproportionately over the last three decades, while skilled workers are a scarce resource (cf. Borjas, 2013, pp. 294–306; Oesch and Rodríguez Menés, 2011; BMWi, 2014). Since projects are executed by humans, multi-project management and human resource management are closely linked to each other. The close link becomes obvious when asking questions like "Which project portfolio can a firm select given its workforce?", "How can workers be assigned to projects such that small project teams result and such that workers are not scattered across a multitude of projects?", or "How can we level the workload over the workforce such that workload is as evenly distributed as possible?" In this thesis, we will present quantitative methods to answer these three questions.

In particular, we are interested in the second question, which asks how to form project teams that are small in relation to project workload and how to avoid scattering of workers across projects. This question came up in a research project triggered by an information technology (IT) center. Though, this question is relevant not only for this IT center but for many firms. Nowadays, almost every firm sets up projects in order to carry out various undertakings. For each project, which is a one-time endeavor, a firm composes a group of employees or workers to accomplish the workload of the project. Such a group of workers is called a project team. Each member of a project team contributes to the project with one or more of his skills. Usually, a worker can be a member of more than one project team at the same time. We and other authors argue that the size of a team should not be too large. A small number of workers per project eases communication within the project team and reduces social loafing and free-riding of team members. If average team size is small, then the average number of assignments per worker to projects is also small. A small number of projects per worker allows workers to better concentrate on each of their projects and avoids productivity losses due to scattering. That is why we aim at small teams. For the very first time, exact and heuristic approaches are presented that help firms with a large multi-skilled workforce and a large project portfolio to form small project teams.

The problems that correspond to our three questions are—directly or indirectly—related to small project teams and will be tackled one by one. The three problems are

M. Walter, *Multi-Project Management with a Multi-Skilled Workforce,* Produktion und Logistik, DOI 10.1007/978-3-658-08036-5_1, © Springer Fachmedien Wiesbaden 2015

(1) the selection of projects under resource constraints, (2) the assignment of multi-skilled workers with heterogeneous skill levels to projects with the objective to obtain project teams of minimum size, and (3) the leveling of workload, i.e., the even distribution of workload across the workforce. Our focus is on the second problem. But before workers can be assigned to projects, projects must be selected. And after project teams have been formed and project workload has been allocated, departmental workload can be allocated such that worker utilization is leveled. Hence, we will tackle these problems in hierarchical order, beginning with the strategic project selection problem at the first stage, continuing with the tactical workforce assignment problem at the second stage, and ending with the operational utilization leveling problem at the bottom stage. Our three-stage top-down approach is partially integrated to ensure feasibility and practical applicability at each stage. The primary goal of this thesis is to find solution methods for the three problems that offer high solution quality within acceptable solution time.

The first problem of our three-stage approach is the selection of a most beneficial set of projects such that the total skill requirements of the selected projects can be accomplished in every period of the planning horizon. We model this project selection problem as a mixed-integer program with a single objective. The objective is to maximize portfolio benefit and we assume that a benefit value is specified for each project. The gist of our model is that it captures resource constraints and ensures a feasible staffing of projects, which is a desirable basis for the second stage. Already Weingartner (1966) and, more recently, Kolisch et al. (2005) have emphasized the necessity of integrating resource constraints into portfolio planning. Our optimization model explicitly considers each worker and his skills. This high resolution is necessary because we distinguish different skill levels, i.e., for each skill we distinguish more experienced from less experienced workers.[1] Yet, the resolution is just high enough to fulfill its purpose; it strikes a balance between the resolution of Yoshimura et al. (2006) and the higher resolution of Gutjahr et al. (2008).

Related to the problem of project selection is the question how skills should be distributed among workers. This question is relevant for making decisions about cross-training of workers. Jordan and Graves (1995) found that a skill configuration called *chaining* is very beneficial, as it can increase the number of feasible project portfolios. For our setting with many workers, we present a skill configuration that outperforms a chaining configuration. We use simulation, an extended version of a flexibility measure of Iravani et al. (2005), and the flexibility measure of Chou et al. (2011) to confirm the effectiveness of the proposed configuration.

The second problem that we consider constitutes the core of this thesis and deals with staffing the selected projects such that small project teams result. The problem is to assign multi-skilled workers to projects and to allocate project workload to workers such that all skill requirements of the projects and all requirements of the departments are satisfied, availabilities of the workers are observed, and the average number of assignments per project, i.e., the average team size, is minimized. We formulate this workforce assignment problem as a mixed-integer program that minimizes the total number of assignments. The formulation takes different skill levels of workers into account: An experienced worker, who has a high skill level, needs less time to accomplish a skill requirement than a worker with a lower skill level.

[1] Even in case of homogenous skill levels, it can be advantageous to explicitly consider each worker compared to aggregating worker capacities, as we will explain in Section 4.2.

Our aim to minimize average team size and the corresponding solution methods are an answer to the call for small teams that has echoed through the literature for a long time. Already in 1975, computer architect and software engineer Frederick Brooks pointed out that growing team size hampers project progress due to increased communication needs. He wrote that "adding manpower to a late software project makes it later" (Brooks, 1982, p. 25), what has become known as Brooks's law. Likewise, Hammer and Champy (1993, p. 144) demanded in their best-selling business book *Reengineering the corporation* that "as few people as possible should be involved in the performance of a process". Especially for multi-project environments, Hendriks et al. (1999) introduced the *project scatter factor*, which measures the number of workers that are assigned to one man-year of project workload. As scattering workload across many workers and scattering workers across many projects is inefficient, Hendriks et al. recommend a small project scatter factor, i.e., small teams. Another drawback of large teams is increased social loafing. Social loafing describes the decrease in individual productivity of workers when they work in a team. Liden et al. (2004) investigated work groups in two U.S. firms and confirmed the hypothesis that the productivity loss increases with team size. They concluded that "organizations will recognize the need ... to keep group size down to a minimum in combating social loafing" (p. 299). So far, these calls for small teams have not been adequately answered by the development of algorithms for complex staffing tasks. We set out to give an answer to this call.

In the literature concerned with workforce assignment problems, Grunow et al. (2004) and Heimerl and Kolisch (2010a) consider problems that have similarities to our problem. The work of Grunow et al. might have come closest to a reply to the call for small project teams. In Section 4.2 of their work, Grunow et al. consider the problem of staffing clinical studies where each clinical study comprises several tasks. A certain number of qualified and available employees must be assigned to each task. As soon as an employee contributes to a task of a study, he is assigned to this study. The objective is to minimize the total number of assignments of employees to studies, i.e., it is desired that an employee devotes his time to a minimum number of studies but contributes as much as possible to the few studies he is assigned to.

There are three crucial points where our approach differs from the approach of Grunow et al. (2004). The first point is that in the model of Grunow et al. the number of employees necessary to accomplish a task is prespecified, whereas in our model the number of workers necessary to cover a skill requirement is not fixed a priori. For example, our model grants the choice to allocate a skill requirement to either one worker who may have to spend 2 weeks on it or to two workers who may require 1 week to accomplish this requirement. The second point is that we take different skill levels into account. This allows us to distinguish an expert who may require only 1 week to accomplish a certain workload from a beginner whom it may take 4 weeks to accomplish the same work. The first two points make our model applicable to a broader range of firms and industries. The third difference is that, although both the problem of Grunow et al. and our workforce assignment problem are NP-hard[2] and thus intractable in case of large-sized instances, Grunow et al. do not outline heuristic solution methods as we do. In our case, the heuristics are necessary to

[2]We prove NP-hardness of our problem in Subsection 4.6.3. Grunow et al. (2004) do not provide complexity results for their problem. In Section 5.2, we show that their problem is NP-hard as well.

solve instances of realistic size, which can comprise more than one thousand workers and hundreds of projects as in the case of the IT center that initiated our research.

The model of Heimerl and Kolisch (2010a) is similar to our model as both models distinguish skill levels and do not prespecify the number of workers necessary to satisfy a skill requirement. The main difference between the models lies in the objective function. Heimerl and Kolisch minimize variable costs that are incurred for each minute of project work performed by employees, whereas we minimize fixed costs or setup costs that are only incurred if an employee contributes to a project, no matter for how long she contributes. Solutions to our model feature small project teams, whereas solutions to the model of Heimerl and Kolisch tend to feature very large project teams. If we aimed at minimizing variable costs, our model would become a linear program (LP). Minimizing fixed costs or average team size, however, cannot be modeled without binary decision variables and is thus a much harder task.

The third problem, utilization leveling, arises when project teams have been arranged and project workload has been allocated to the team members. After planning at the second stage has been finished, the resulting distribution of project workload may have led to an unbalanced utilization of workers. Hence, when workers are compared with respect to their workload, discrepancies may occur. Then, departmental workload can be distributed such that utilization is leveled as well as possible. This utilization leveling problem is modeled as an LP at the third stage of the planning hierarchy.

Equal loading of workers is valuable for firms and workers for at least two reasons. First, an equal utilization is deemed fair by workers and increases worker satisfaction (cf. Lee and Ueng, 1999). Second, a leveled load distribution facilitates a simple and popular working time organization where working time is constant in each period (cf. Huang et al., 2006). Leveling avoids overtime and prevents underutilization during regular working time. Solving the utilization leveling problem completes the three-stage approach outlined in this thesis.

The main contribution of our work is that we develop and identify methods to solve the three outlined problems. In particular, we outline construction heuristics to compose project teams of minimum average size. One of the proposed heuristics, a drop heuristic, applies in one of its variants the generalized network simplex method. For this simplex method, we provide detailed pseudo code, which, to the best of our knowledge, has not been published so far.

Our thesis makes further contributions that are relevant from a theoretical and a practical perspective. With regard to practice, we tackle three problems that have gained increased relevance for firms. For each problem, we present a mathematical model and discuss limitations of the model that should be observed by practitioners. For the workforce assignment problem, we outline a lean model that requires a relatively small amount of input data. The model gets along without cost data, which are often difficult to estimate. With regard to theory, we analyze the complexity of our three problems and classify them in terms of their computational tractability.

The key results of our work can be summarized as follows. The project selection problem that we consider at the first stage is NP-hard in the strong sense, but even instances that represent cases of large firms can be handled well by state-of-the-art branch-and-cut solvers for mixed-integer programs. The workforce assignment problem at the second stage is also strongly NP-hard. Only for small-sized instances, solutions of acceptable quality

can be determined within reasonable time. From the heuristics that we developed, a variant of a drop method that uses a commercial LP solver for subproblems performs best. The drop method profits from temporal decomposition, problem-specific lower bounds, and other problem-specific devices. The utilization leveling problem arising at the third stage can be modeled as an LP and is thus solvable in polynomial time. For this problem, we developed a specially tailored algorithm that outperforms a commercial state-of-the-art LP solver, which applies the simplex method. A further notable result was obtained with regard to skill configurations. In line with recent work, we found that a configuration which follows the chaining principle is inferior to a more diverse skill structure.

Our work has implications for both research and practice. With respect to research, we make a novel contribution to the growing field of multi-project planning. We satisfy a long outstanding demand by providing methods for building small project teams. Our methods are a first landmark and can serve as a basis for further research. With respect to practice, our methods can give valuable support to managers who are charged with project portfolio and human resource planning. Our methods help them in three ways. First, our approach helps managers to select beneficial portfolios which fully exploit workforce potential but which do not overstrain any workforce member. Second, our approach helps to form small project teams, which raise the chance of smooth project execution. Additionally, the approach avoids scattering workers across projects and thereby decreases harmful stress and increases worker satisfaction. Third, the approach helps to equally distribute workload such that the workers feel that they are treated fairly. Altogether, our methods can improve the well-being of a firm and its workers alike.

The remainder of this thesis is organized as follows. In **Chapter 2**, we provide some essentials of multi-project planning and background information about multi-skilling, e.g., about measuring skill levels, which are required input for our approach. Furthermore, we explain the motives for our approach: We stress why resources should be taken into account for composing a project portfolio and we present theoretical, experimental, and empirical findings which suggest that small project teams and a small number of projects per worker are favorable. In **Chapter 3**, we describe our three problems and introduce the notation, i.e., we introduce symbols for sets, parameters, and variables that define our problems. Mathematical optimization models for these problems are presented in **Chapter 4**. In this chapter, we also discuss limitations of the models and classify the hardness of our problems according to complexity theory. Work that is related to our problems and models is reviewed in **Chapter 5**.

Solution approaches to our problems are outlined in **Chapter 6**. For the project selection problem, the branch-and-cut method offered by the solver package CPLEX provides good solutions. For the workforce assignment problem, we present valid inequalities that tighten the corresponding model and four heuristic solution methods. We also give a detailed description of the generalized network simplex method, which is integrated into one of the heuristics. For the utilization leveling problem, we outline a polynomial-time algorithm. **Chapter 7** deals with thorough testing of the proposed solution methods. We describe the generation of test instances and report on the results of our performance analysis. The results of the analysis and their implications are discussed in **Chapter 8**. **Chapter 9** concludes this thesis with a summary and an outlook.

Chapter 2

Background and motivation

In this chapter, we provide general information on three essential elements of the environment in which our three problems are embedded. These elements are multi-project management, multi-skilled workers, and teamwork. In Section 2.1, we elaborate on multi-project management, which is concerned with managing a portfolio of parallel undertakings. We emphasize the importance of multi-project management, explain its main tasks, and give an overview of contributions that address the main tasks. Two of these main tasks, namely, project selection and staffing, are part of our approach. In Section 2.2, we turn to multi-skilled workers. Here, we describe advantages of cross-training and give a brief overview of methods to determine skill levels, which are required as input data for our models. Furthermore, we describe the flexibility design problem that is associated with a multi-skilled workforce. Later, we make a novel contribution to this design problem in Section 7.2. Eventually, we address teamwork in Section 2.3 where we give reasons why small teams are advantageous.

2.1 Multi-project management

In this section, we briefly review principles of multi-project management. First of all, we stress the importance of multi-project management. Then, we elaborate on its three main planning tasks, which are portfolio selection, scheduling, and staffing of projects (cf. Heimerl and Kolisch, 2010a, p. 344). All three tasks are affected by the availabilities and capabilities of (human) resources whose crucial role is emphasized throughout this section.

The importance of multi-project management has increased over the last decades and is still growing. In the middle of the last century, project and multi-project management gained momentum; the share of project work has increased since then and the penetration of firms by corresponding management methods has not stopped at the beginning of this century (cf. Kerzner, 2013, pp. 47–63). The ongoing growth of multi-project management and its still increasing relevance is reflected in the currently growing need for portfolio and project management software (cf. Figure 2.1).

The rise in project work has been a response to several developments. Accelerated technological progress and faster changing consumer needs have led to shorter product life cycles and have rewarded shorter product development times. This in turn put time pressure on organizations and caused shorter life spans of the respective organizational structures. Intensified national and international competition rewarded customer orientation, i.e., customized and thus unique solutions. Shorter life spans of organizational

M. Walter, *Multi-Project Management with a Multi-Skilled Workforce,* Produktion und Logistik, DOI 10.1007/978-3-658-08036-5_2, © Springer Fachmedien Wiesbaden 2015

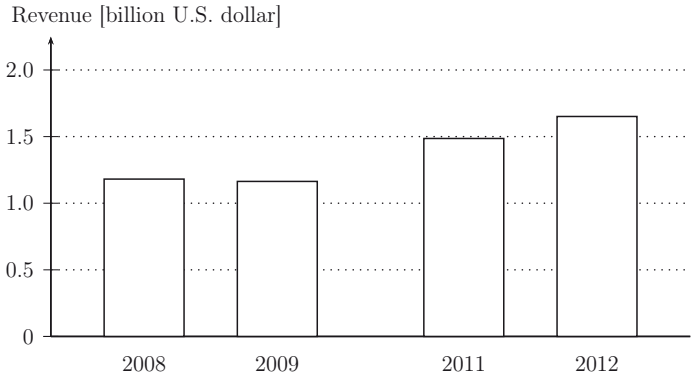


Figure 2.1: Global revenue from project and portfolio management software from 2008 to 2012 (data provided by Statista (2013), original source: Gartner, Inc.)

structures and customer orientation favor project work, which is temporary and unique by nature. The increase in project work has amplified the importance of multi-project management, which is concerned with planning, executing, and controlling. In the realm of planning, the major management tasks are selecting, scheduling, and staffing projects. We elaborate on these main tasks because they overlap with our approach.

The first main task of multi-project management is selecting projects. The resulting project portfolio contains both external and internal projects. External projects originate outside a firm, whereas internal projects come from within the firm. Examples are a customer order and a process improvement project, respectively. Common criteria for project selection are strategic fit, urgency, and profitability of a project amongst others (cf. Pinto, 2010, pp. 90–125; Rüdrich, 2011). The strategic fit expresses how well the project contributes to overall long-term goals of the organization, whereas urgency relates to the operational need for the deliverables of the project. The criterion profitability measures the economic advantage of the project. In our approach, we condense information emerging from several criteria into a single benefit value for a project.

Two variants of project selection problems are distinguished: static and dynamic project selection. The static variant, which we consider in our approach, considers a set of candidate projects from which only a proper subset can be selected for implementation. A decision is made only once. If resource constraints are taken into account, this variant is related to the knapsack problem where a subset of items must be chosen such that their total weight does not exceed the capacity limit of a knapsack and their total utility is maximized. The dynamic variant of the project selection problem considers the situation where project proposals arrive over time (cf. Herbots et al., 2007). Every time when a project arrives, a decision has to be made whether the project is selected or not.

When such a decision must be taken, details of future arrivals, e.g., resource requirements, benefits, and arrival times of the projects arriving next, are not known.

Static project selection problems are formulated either as single- or multi-period problems. In the case of multi-period formulations, the planning horizon is divided into periods. Hence, the temporal process of a project can be taken into account. A multi-period approach is advantageous especially if scarce resources are considered because the schedule of a project—whether already fixed or still flexible—does not result in a constant load over time for all resources in general. The following example demonstrates the advantage of a multi-period model when resource requirements vary over time.

Example 2.1 Let two candidate projects that last one year each have a resource demand for 4 man-years while 8 workers are available. Then, both projects can be selected in a single-period model as they do not violate resource constraints. However, if each project requires 6 full-time workers in the first half of the year and only 2 workers in the second half, it is not possible to implement both projects given 8 workers. Nevertheless, the portfolio containing both projects would be a feasible portfolio in the single-period model.□

Since changes in resource requirements over time, e.g., the dynamic requirements stated in Example 2.1, cannot be captured by a single-period model, we opted for a multi-period formulation. A multi-period model allows to take non-constant resource demands and supplies into account.

In the literature on project selection, however, an explicit consideration of resource requirements and resource availabilities is not a given and often, resources are neglected. Textbooks on project management frequently suggest selection procedures which do not consider resource constraints (cf. GPM, 2008, pp. 447–467; Pinto, 2010, pp. 90–125). Of course, procedures that do not account for resources can be justified, as in some situations resource constraints are not present or because the procedures have another focus. In many situations, though, resources are scarce, and neglecting resource constraints can lead to plans that cannot be implemented.

Many approaches to project selection consider budget constraints, i.e., limited financial resources. Financial requirements of projects may include cost for personnel. Some approaches that consider only budget constraints but no other resource constraints may imply the assumption that workers with the necessary qualification profiles can just be bought. However, such an assumption is risky. Since skilled workers are rather scarce, this assumption is questionable and may lead to failure when projects are carried out.

Table 2.1 gives an overview of contributions to static project selection problems that we reviewed for this thesis. Some of these contributions will be treated in more detail in subsequent chapters, especially in Section 5.1. Table 2.1 distinguishes single-period from multi-period models and whether only budget restrictions or resource constraints including potential budget restrictions are considered. Furthermore, we marked contributions that consider skills, be it in the form of multi-skilled resources or in the form of different skill levels of mono-skilled resources. Some of the listed publications integrate scheduling and/or staffing and some explicitly consider multi-skilled resources. Scheduling is integrated into the models of Chen and Askin (2009), Escudero and Salmeron (2005), Ghasemzadeh et al. (1999), and Kolisch et al. (2008). Kolisch et al. (2005) and Taylor et al. (1982) consider problems with a time-resource trade-off where project start times are fixed but project durations vary with the amount of allocated resources. Both scheduling

and staffing is incorporated in the selection models of Gutjahr et al. (2008, 2010). Lopes et al. (2008), Yoshimura et al. (2006), and Taylor et al. (1982) integrate selection and staffing.

Table 2.1: Classification of selected publications on static project selection procedures

		Constraints		
	Multi-period model	Only budget	Resources (incl. budget)	Skills
Fox et al. (1984)		•		
Golabi et al. (1981)		•		
Gurgur and Morley (2008)		•		
Lai and Xue (1999)		•		
Lopes et al. (2008)		•		•
Muralidhar et al. (1990)		•		
Schmidt (1993)		•		
Tavana (2003)		•		
Eilat et al. (2006)			•	
Santhanam et al. (1989)			•	
Santhanam and Kyparisis (1996)			•	
Yoshimura et al. (2006)			•	•
Chen and Askin (2009)	•		•	
Doerner et al. (2004)	•		•	
Escudero and Salmeron (2005)	•		•	
Ghasemzadeh et al. (1999)	•		•	
Graves and Ringuest (2003)	•		•	
Gutjahr et al. (2008)	•		•	•
Gutjahr et al. (2010)	•		•	•
Kolisch et al. (2005)	•		•	
Kolisch et al. (2008)	•		•	
Taylor et al. (1982)	•		•	
Weingartner (1966)	•		•	

Skills are considered in only four of the listed works. In Yoshimura et al. (2006) and Gutjahr et al. (2008, 2010), multi-skilled workers constrain the composition of the project portfolio. Skill levels are static in the model of Yoshimura et al. (2006), whereas Gutjahr et al. (2008, 2010) take learning and forgetting into account resulting in dynamic skill levels that increase through exercising the respective skill and decrease when the skill is not used. In Lopes et al. (2008), projects are selected and assigned to students who must complete a project as part of their curriculum. Here, students exhibit different average grades that can be interpreted as distinct skill levels. A soft constraint requires that the mean skill of each project team is not below a minimum level.

In all contributions, portfolio benefit or value is maximized while occasionally other goals are pursued in addition to benefit maximization. Some authors describe how multiple criteria can be aggregated, whereas Yoshimura et al. (2006), Doerner et al. (2004),

Graves and Ringuest (2003), and Gutjahr et al. (2010) consider genuine multi-criteria models with a separate objective function for each criterion. Apart from the objective of maximum portfolio benefit, three models feature additional objectives that refer to skills and worker satisfaction. Lopes et al. (2008) minimize the total deviation of all project teams from the minimum mean skill level. Yoshimura et al. (2006) pursue the aim of maximizing the skill supply for each project and maximizing worker satisfaction. Gutjahr et al. (2010) seek for solutions that lead to an optimal growth in skill levels due to learning.

The second main task of multi-project management is project scheduling, which is concerned with determining a start time for each project and sometimes also for each activity of a project if there is leeway. Start times must be chosen such that resource constraints and temporal constraints are regarded. Since resources are required to perform project activities, a feasible schedule must observe availabilities of renewable resources such as manpower and of non-renewable resources such as budget. Temporal constraints can comprise minimum and maximum time lags between project activities. Frequent objectives for scheduling are maximization of the net present value, minimization of costs, and minimization of project duration, also called makespan minimization. Though, Tukel and Rom (1998) found in a survey among U.S. project managers from various industries that quality was the most important scheduling objective for the respondents; it was more important than time and cost. However, a precise definition of the term quality was not given by Tukel and Rom.

In general, a multi-project scheduling problem can be transformed into a single-project scheduling problem for which a rich body of literature exists that provides different solution approaches for a large number of objective functions (cf. Neumann et al., 2003, for example). To sketch the transformation, assume that each project is represented by an activity-on-node network whose nodes correspond to project activities and whose arcs correspond to temporal constraints. Then, the separate project networks can be merged into one network that represents a meta-project and contains all projects as subprojects. In this case, multi-project and single-project scheduling problems are equivalent.

For the project selection and staffing problem that we consider in this thesis, we assume that the schedule of each project is already fixed prior to the decision about the project portfolio. This assumption holds true for many situations nowadays. Due to increased competition, clients can often demand tight due dates that do not give the freedom to shift activity start times (cf. Kolen et al., 2007, pp. 530–531). Nevertheless, it is worthwhile to explore scheduling approaches because they often integrate staffing decisions and sometimes explicitly consider multi-skilled workers.

When multi-skilled workers are considered, it is possible but not advisable to represent a project scheduling problem as a multi-mode resource-constrained project scheduling problem (MRCPSP). In the MRCPSP, activities can be executed in alternative modes that differ in resource usage and activity duration (cf. De Reyck and Herroelen, 1999; Heilmann, 2003). To give an example, consider an activity that can be executed in two modes. In the first mode, the activity is executed by an expert and finished within two days. In the second mode, two beginners perform the activity and need three days. To demonstrate the disadvantage of a multi-mode model, consider a workforce of 20 workers and an activity that can be accomplished by any of the 20 workers alone but also by any team formed by the workers. Each distinct, non-empty subset of workers may represent

a mode in which the task can be performed. Then, the activity can be executed in $2^{20} - 1 = 1\,048\,575$ different modes.[1] Here, the size of a model that explicitly describes all possible modes increases exponentially in the number of workers. Hence, if each worker is considered as an individual resource and workers have overlapping skill sets, the number of alternative modes per activity usually is so large that a multi-mode representation is not adequate (cf. Bellenguez-Morineau and Néron, 2007, pp. 156–158). That is why multi-mode models are not common for problems with multi-skilled workers. The work of Tiwari et al. (2009) is an exception as they use an MRCPSP formulation for scheduling activities given a multi-skilled workforce with heterogeneous skill levels.[2]

Table 2.2 lists a choice of papers that tackle either multi-project or single-project scheduling problems or problems that are no project scheduling problems but scheduling problems that involve multi-skilled workers. Papers that explicitly refer to multi-project scheduling are marked in the table. Furthermore, the table indicates when contributions consider multi-skilled resources, when skill levels are distinguished, and what kind of objective is pursued. Most of the listed papers that deal with multi-project scheduling determine project start times only and assume that activity start times are coupled to the start time of the corresponding project and cannot be shifted. Solely Kolisch and Heimerl (2012) determine project and also activity start times. In the models of Gutjahr et al. (2008, 2010) and Wu and Sun (2006), there is also freedom: The workload of an activity must be accomplished sometime between the release and the due date of the activity. In Kolisch et al. (2008), a start time and a mode must be chosen for each project; different modes are associated with different project durations and different resource requirement profiles.

Time related and cost related objectives are very common for scheduling approaches, as Table 2.2 reveals. In Alfares and Bailey (1997), costs depend on project duration and on the number of workers that are deployed in different days-off tours in each week of the planning horizon. Barreto et al. (2008) suggest different objectives for their scheduling and staffing problem, e.g., the minimization of project duration, staffing costs, and team size. Li and Womer (2009a) try to schedule tasks on a naval ship such that the size of the required crew is minimized. In Li and Womer (2009b), costs of a project schedule directly depend on the number of workers that are required to staff the project; in a numerical study, the costs of assigning a worker to the project are the same for each worker, hence they minimize project team size in the numerical study. Valls et al. (2009) consider three objectives, which are lexicographically ordered; their objective with lowest priority is leveling the workload of employees, which is also our lowest-priority objective. Dodin and Elimam (1997) take switching costs into account that are incurred every time when an employee is assigned to a task which belongs to another project than his previous

[1] The number of modes is even higher if a team can accomplish the task in different modes, as it will be the case in our models.

[2] The work of Tiwari et al. (2009) is noteworthy for a second reason as the authors explicitly consider quality in their approach. They require that an activity is completed by a suitable worker with a certain skill level in order to meet a desired quality requirement. However, a less skilled worker can start to process the activity and hand it to a worker with the required skill level for the final touch. As workers with high skill levels are a bottleneck, this splitting of an activity's accomplishment can shorten project duration.

Table 2.2: Classification of selected publications on (project) scheduling

		Skills		
	Multi-project model	Multi-skilled ressources	Different skill levels	Aim*
Alfares and Bailey (1997)				C
Drexl (1991)			•	C
Bellenguez-Morineau and Néron (2007)		•		T
Correia et al. (2012)		•		T
Drezet and Billaut (2008)		•		T
Ho and Leung (2010)		•		O
Li and Womer (2009a)		•		C
Li and Womer (2009b)		•		O
Vairaktarakis (2003)		•		T
Barreto et al. (2008)		•	•	C,T,O
Cordeau et al. (2010)		•	•	T
Fırat and Hurkens (2012)		•	•	T
Hegazy et al. (2000)		•	•	T
Valls et al. (2009)		•	•	O
Chen and Askin (2009)	•			B
Escudero and Salmeron (2005)	•			B
Ghasemzadeh et al. (1999)	•			B
Kolisch et al. (2008)	•			B
Grunow et al. (2004)	•	•		C
Bassett (2000)	•	•	•	C
Dodin and Elimam (1997)	•	•	•	C
Gutjahr et al. (2008)	•	•	•	O
Gutjahr et al. (2010)	•	•	•	O
Heimerl and Kolisch (2010a)	•	•	•	C
Kolisch and Heimerl (2012)	•	•	•	C
Wu and Sun (2006)	•	•	•	C

*Abbreviations for aims: B = portfolio benefit, C = cost,
T = time (makespan, completion times, lateness), O = other

task.[3] In Wu and Sun (2006), it is possible to outsource workload to external workers in order to meet due dates; the aim is to minimize costs for external staff.

The listed works do not only differ in the characteristics stated in Table 2.2. Another difference is, for example, whether the number of workers necessary to accomplish a task is prescribed or not. A further difference in the respective models is whether a worker can contribute to one or more tasks per period. Moreover, the length of the planning horizons differs. A schedule searched for can span one day (cf. Ho and Leung, 2010; Valls et al., 2009; Grunow et al., 2004), a couple of days (cf. Drezet and Billaut, 2008; Cordeau et al., 2010; Fırat and Hurkens, 2012), several weeks (cf. Barreto et al., 2008; Li and Womer,

[3]The model presented by Dodin and Elimam (1997) is not utterly correct, as the constraints that are intended for counting the occurrences of switching can actually exclude some feasible solutions.

2009a; Hegazy et al., 2000), a number of month (Alfares and Bailey, 1997; Wu and Sun, 2006, cf.), one or two years (cf. Bassett, 2000; Gutjahr et al., 2008, 2010), or even a period between three and five years (cf. Escudero and Salmeron, 2005; Ghasemzadeh et al., 1999; Kolisch et al., 2008).

Teams are addressed by five works listed in Table 2.2. Ho and Leung (2010) solve a crew pairing and routing problem where teams of two persons must be formed and a job sequence must be assigned to each team such that at least one team member masters the skill that is required by a job in the sequence. For a scheduling and staffing problem, Barreto et al. (2008) formulate alternative objective functions that lead the search to either a fastest, a least expensive, a best qualified or a smallest team, for example. Cordeau et al. (2010) and Fırat and Hurkens (2012) consider a problem where for each day technicians must be grouped in teams that stay together for one day. A team can only accomplish tasks whose skill requirements can be satisfied by the team. In Grunow et al. (2004), workers that are assigned to tasks which belong to the same clinical study form a team. In the corresponding staffing subproblem that is embedded in their scheduling approach, Grunow et al. minimize average team size as we do in our approach. In our approach, though, the number of workers that are needed to accomplish a task of a project is not fixed a priory. Furthermore, we distinguish different skill levels for workers and do not assume homogeneous efficiencies.

Most of the listed papers on scheduling integrate staffing, i.e., tasks are assigned to individual workers or to single teams. In contrast, Alfares and Bailey (1997), Hegazy et al. (2000), Chen and Askin (2009), Escudero and Salmeron (2005), Ghasemzadeh et al. (1999), and Kolisch et al. (2008) only ensure resource feasibility of their schedules but do not specify which resource has to accomplish which tasks.

The third main task of multi-project management is to assign workers to projects and allocate project workload to them. For this task, many aspects such as availabilities, skills, and capacity for teamwork must be taken into account: Availabilities of workers must be observed, especially if workers have other duties apart from project work. Other duties are common in matrix organizations, where long-term organizational structures, which are embodied by functional departments, for example, and short-term project organizations coexist. Here, both department and project managers demand workers and hence coordination is necessary (cf. Schneider, 2008). Workers should master the skills that are needed to successfully accomplish those tasks they are assigned to. If collaboration of project team members is essential, an employee's ability to work in a team and his capability to cooperate with each co-worker in the team should be regarded. Our staffing model explicitly considers the situation in a matrix organization and takes workloads of functional departments and projects into account. Skills of workers and their efficiencies are also regarded.

In the literature, various aspects have been considered in staffing models and many approaches to workforce assignment problems have been presented. Table 2.3 contains a choice of papers that deal with staffing for the day-to-day business, staffing of single projects, or staffing of multiple projects. Those papers that consider multiple projects are marked in the second column of Table 2.3. The table further distinguishes whether mono- or multi-skilled workers are considered, whether workers differ in skill levels, and whether a single- or multi-period problem is tackled.

There are many other differences between the works compiled in Table 2.3. These

Table 2.3: Classification of selected publications on staff assignment

		Skills		
	Multi-project model	Multi-skilled ressources	Different skill levels	Multi-period model
Miller and Franz (1996)				•
Corominas et al. (2005)		•		
Gomar et al. (2002)		•		•
Krishnamoorthy et al. (2012)		•		•
Kumar et al. (2013)		•		
Slomp and Molleman (2002)		•		
Valls et al. (1996)		•		•
Brusco and Johns (1998)		•	•	•
Campbell and Diaby (2002)		•	•	
Eiselt and Marianov (2008)		•	•	
Fowler et al. (2008)		•	•	•
Otero et al. (2009)		•	•	
Fitzpatrick and Askin (2005)	•			
LeBlanc et al. (2000)	•			•
Lopes et al. (2008)	•		•	
Reeves and Hickman (1992)	•		•	
Certa et al. (2009)	•	•	•	
Patanakul et al. (2007)	•	•	•	
Santos et al. (2013)	•	•	•	•
Yoshimura et al. (2006)	•	•	•	

differences concern the level of detail with which workers are modeled, the interpretation of skill levels, and the objectives pursued, for example. First of all, the listed papers consider either individual workers or groups of workers where the workers of each group have identical skill sets and identical skill levels. Such groups of homogeneous workers are found in Corominas et al. (2005), Valls et al. (1996), Brusco and Johns (1998), and Fowler et al. (2008).

A major difference lies in the interpretation of skill levels. Those papers that do not only distinguish whether a worker masters a certain skill or not but that also distinguish the level of excellence with which a skill is mastered, interpret skill levels in various ways. Some authors, namely, Brusco and Johns (1998), Fowler et al. (2008), and Otero et al. (2009), interpret skill levels as a measure of efficiency. We follow the same interpretation. According to this interpretation, a worker with a higher skill level accomplishes a corresponding task faster than a less skilled worker. Other authors, videlicet, Campbell and Diaby (2002), Lopes et al. (2008), Reeves and Hickman (1992), Certa et al. (2009), and Patanakul et al. (2007), interpret skill levels as a measure of quality. Here, utility of a task completion is the higher, the higher the skill level of the responsible worker is. Yoshimura et al. (2006) and also Barreto et al. (2008) (see Table 2.2) interpret skill levels in terms of both efficiency and quality, whereas Eiselt and Marianov (2008) use the positive difference between a worker's skill level and the skill level demanded by a task to express the degree

of boredom resulting from the corresponding assignment for the worker. In Santos et al. (2013), the skill level impacts training costs that are incurred if a worker whose skill level is lower than the maximum level is assigned to a task requiring the skill. In the approach of Otero et al. (2009) a low skill level can be compensated by high levels in related skills.

As in the case of papers on scheduling, the staffing models differ with respect to the maximum number of projects or tasks a worker can contribute to in each period or during the complete planning horizon. The staffing models differ also with respect to the number of skills required by a task and whether the number of workers required to perform a task is predefined or not. Planning horizons range between one day (cf. Krishnamoorthy et al., 2012; Brusco and Johns, 1998; Campbell and Diaby, 2002), a couple of weeks (cf. Gomar et al., 2002), several months (cf. Eiselt and Marianov, 2008; Fowler et al., 2008; Patanakul et al., 2007), and one year (cf. LeBlanc et al., 2000). Most authors assume for their models that the workforce is fixed and that its size cannot be altered, i.e., that capacity is fixed. However, Gomar et al. (2002) and Fowler et al. (2008) include decisions about hiring and laying off workers; Santos et al. (2013) allow for hiring but not for firing. In Eiselt and Marianov (2008), the workforce is fixed but tasks can be outsourced.

Objectives of the listed contributions are different as well. Some models search for an assignment that maximizes the preferences of the workers, some minimize staffing costs. Staffing costs can comprise several types of cost, e.g., costs for hiring, firing, overtime, outsourcing, and training. The goal of a minimum number of required workers is pursued by Krishnamoorthy et al. (2012), Valls et al. (1996), and Brusco and Johns (1998); though, none of them considers a multi-project environment. Other objectives apart from the aforementioned are pursued as well. Multi-criteria models with more than one objective function are formulated by Reeves and Hickman (1992), Certa et al. (2009), and Yoshimura et al. (2006).

A special problem is the assignment of project managers to projects. The papers of LeBlanc et al. (2000) and Patanakul et al. (2007) address only this problem, while Yoshimura et al. (2006) tackle this problem as one of many. Patanakul et al. (2007) explicitly consider a manager's effort to head more than one project team simultaneously. Time needed for switching from one project to another is taken into account. This switching time represents the productivity loss that arises due to multi-tasking. Furthermore, the maximum number of projects a manager can lead is limited in the model of Patanakul et al.

Team issues are addressed by six contributions in Table 2.3. Certa et al. (2009) consider the relationship between pairs of workers and assume that a parameter describes for each pair of workers their preference to work together. One of the goals pursued by Certa et al. is to form project teams such that the preferences for cooperation of those team members who perform the same skill are maximized. Similarly, Kumar et al. (2013) seek for an assignment of workers to tasks such that those tasks whose execution requires cooperation between the responsible workers are assigned to workers who can collaborate as well as possible. Likewise, Yoshimura et al. (2006) integrate mutual compatibility of team members into one of their objective functions. Fitzpatrick and Askin (2005) try to determine a team such that personality traits and instinctive behaviors of team members are optimally balanced. Lopes et al. (2008) and Reeves and Hickman (1992) tackle problems where students must be assigned to projects that are part of the students' curricula. In both contributions, one aim is to form project teams of equal quality. Team

quality is defined by the mean of the team members' average grades, i.e., by the mean skill level of the students belonging to the team.

In two contributions, the proposed computer-based approaches were compared to manual planning approaches. These comparisons revealed that even small-sized instances can ask too much of a manual planner and that human intuition can be misleading. Barreto et al. (2008) (see Table 2.2) conducted an experiment where test persons had to solve small-sized instances of their problem without the support of computer-based solution methods. Most of the participants did not find an optimal solution and spent much more time on the task than the automated solution method that Barreto et al. have outlined. Otero et al. (2009) asked test persons to select the most suitable worker out of a workforce for a task with given skill requirements. It came out that most test persons attached great importance on the main skill, i.e., on the skill for which—among all skills demanded by the task—the highest level was required. They based their decision on this main skill and neglected the remaining skills required by the task and mastered by the workers. All in all, both comparisons showed that manual planning for complex assignment problems is tedious and tends to result in suboptimal assignments.

From the reviewed publications in Tables 2.1–2.3, it becomes obvious that there are manifold aspects that should be carefully considered by those who are responsible for multi-project management. One of these aspects is team size. So far, team size has not been considered for a multi-project environment with multi-skilled workers and heterogeneous skill levels. We set out to close this gap.

Apart from the three main tasks of the planning stage, there are other important tasks in the realm of multi-project management, e.g., risk management or actual project execution. During project execution, managers must motivate workers to reach their desired performance levels, resolve interpersonal conflicts that often accompany teamwork, and ensure that the project is completed on time and in budget and that the deliverables of the project meet quality requirements.

Our approach covers only some tasks out of the wide range of tasks associated with project portfolio management. However, our models for project selection and workforce assignment are crucial parts of the planning process, which have great impact on subsequent stages such as project execution. To give an example of this impact, imagine a project team that was formed but cannot meet the requirements of a project due to lack of time or lack of skills. Workers and their skills are an essential part of our models and are considered in more detail in the next section.

2.2 Multi-skilled workers and flexibility design

At the beginning of Chapter 1, we pointed to the importance of human resource management and its close link to multi-project management. In the first part of this section, we will underscore the close relationship and explain why human resource management plays a crucial role for a firm's performance. The human resources that we consider are multi-skilled workers with heterogeneous skill levels. The second part of this section will bring them into focus: We define the term *skill* and give an example for a multi-skilled worker; we refer to methods with which information about skill levels can be obtained and made accessible, because skill levels are necessary input data for our models; eventually,

advantages and disadvantages of multi-skilling, which is also called cross-training, will be briefly discussed. A main advantage of multi-skilling is greater flexibility, which can be exploited when the demand for two or more skills varies over time. This advantage gives rise to the question how each worker should be cross-trained or, put differently, which skill should be mastered by which worker. This question is known as the flexibility design problem. The third and final part of this section elaborates on this problem. We introduce a prominent solution to the problem: a skill configuration termed *chain*. Furthermore, we outline simple but effective measures to estimate the quality of a skill configuration. These measures, which have been proposed only recently, are used later in Section 7.2 to compare different skill configurations and to derive a refined measure.

When workers are assigned to projects, the realm of human resource management is affected. Human resource management is responsible for assigning tasks to workers, for recruiting, for training and compensating workers, but also for ensuring health and satisfaction of workers. The overall goal of human resource management is to provide workers with required behavior and skills such that these workers can achieve the goals set by the firm, e.g., the successful execution and completion of projects. The importance of human resource management has grown over the last decennia, especially in developed countries, due to increased demand for skilled workers and due to demographic change.

A shift in labor demand has made the recruitment of project workers more difficult. Compared to middle- and low-skill occupations, the relative employment share of high-skill occupations has experienced a massive increase during the last three decades; wages for high-skill jobs have risen disproportionately. Several factors may have contributed to this development (cf. Borjas, 2013, pp. 294–306; Autor et al., 2008; Goos et al., 2009; Dustmann et al., 2009; Oesch and Rodríguez Menés, 2011; Autor and Dorn, 2013). Two persuasive explanations, which are underpinned by empirical data, are the hypothesis of skill-biased technological change (cf. Berman et al., 1998) and the hypothesis of task-biased technological change (cf. Autor et al., 2006, 2008; Spitz-Oener, 2006; Goos et al., 2009; Oesch and Rodríguez Menés, 2011; Autor and Dorn, 2013). The former hypothesis suggests that technological change acts as a complement for skilled labor and as a substitute for unskilled labor. The hypothesis of task-biased technological change, which is also termed routinization hypothesis, argues that computerization and automation have had a negative impact on middle-skill occupations characterized by cognitive or manual *routine tasks* and a positive impact on high-skill and low-skill jobs, which are associated with abstract *non-routine tasks* and manual *non-routine tasks*, respectively. Since project work is usually characterized by non-routine tasks and tends to require skilled labor, human resource managers are exposed to severe competition when seeking for suitable workers and must fear that current employees are headhunted.

Demographic change exacerbates the difficulties in recruiting in some countries, e.g., in Germany. In these countries, a long-term decline of the fertility rate below the replacement level has led to a decrease in the number of people in working age, while a long-run increase in life expectancy has prevented labor demand from falling. Hence, skilled workers have become a scarce resource in some occupations, particularly in nursing and in occupations related to mathematics, informatics, natural science, and technology (cf. BMWi, 2014; Demary and Erdmann, 2012).

The shortage of skilled labor ascribes human resource management an important role. Firms must deploy their workers effectively and efficiently. An efficient use of labor

is facilitated when workers are multi-skilled. Furthermore, firms must ensure that a certain level of worker satisfaction is attained. Satisfaction can be secured not only by fair remuneration but also by fair workload allocation which does not overstrain single workers. In consequence, a firm must take the interests of workers, their capabilities and skills, and their availabilities into account. Then, short-term success, e.g., smooth project execution, and long-term success, e.g., low labor turnover, become more likely.

In the second part of this section, we concentrate on skills and multi-skilling. In our context, the term *skill* refers to a worker's knowledge in a certain field and to her ability to perform a certain set of tasks. We consider the term *competency* as a synonym for skill. Both stand for a potential that becomes visible through performance, i.e., when knowledge and abilities are successfully applied to accomplish a task or to solve a problem (cf. Gnahs, 2010, pp. 19–24; Erpenbeck and Rosenstiel, 2007, pp. XVII–XXI; Shippmann et al., 2000, pp. 706–707). To give an example for skills, consider a worker in the IT center of an insurance firm. His skills may comprise the ability to write computer programs, the qualification to act as a system administrator, and knowledge of processes in life insurance business. Since this worker masters more than one skill, he is an example for a *multi-skilled worker*. A worker who masters only one relevant skill is termed a *mono-skilled worker*.

The term skill refers in our work also to tasks and projects. Both ask for skills in order to be accomplished. Hence, tasks and projects demand workers who provide the required skills. A project of the IT center may be the setup of a new database for recording information about life insurance customers and their contracts. For this project, three skills may be needed, namely, familiarity with life insurance business, knowledge about database design, and acquirements in database management.

The level of detail with which skills are distinguished by a firm depends on its projects and its workforce. In an IT center it might be reasonable to differentiate between programming skills in two languages such as C++ and Java. In a small company, however, where only a small share of projects asks for software applications, where these programs base on rather simple code, and where only a few workers have programming skills, a broader skill category that only records whether a worker has programming experience or not may be appropriate. In this case, it is assumed that a worker with programming experience in one programming language can acquaint herself with any other programming language and can take over any programming task.

Skills that are relevant for accomplishing project tasks must be carefully identified. Relevant skills comprise hard skills, i.e., professional and methodological competencies, as well as soft skills, e.g., social competencies. Our work focuses on professional skills, though the following remarks are valid for hard and soft skills. The vantage point for skill identification are the projects, which represent the demand side. Techniques from *job analysis* and *competency modeling* can be used to identify relevant skills. Both methods, job analysis and competency modeling, are used to elaborate job descriptions that support recruitment and worker assignment. Job analysis (cf. Flanagan, 1954; Lopez et al., 1981; Levine et al., 1983, 1988) tends to have a short-term, task-oriented focus on matching workers and jobs, whereas competency modeling (cf. Mansfield, 1996; Shippmann et al., 2000) is rather long-term and worker-oriented in its nature and is geared towards matching workers and organizations. Though, the overlap area of both methods is large (cf. Schneider and Konz, 1989; Shippmann et al., 2000).

When all relevant skills have been recognized on the demand side, a firm has to identify for each skill those workers that supply the respective skill. Since workers can differ in efficiency with respect to a skill, identification not only of workers mastering a skill but also of the corresponding efficiencies, i.e., of the skill levels, is necessary for making informed and reasonable decisions on project assignment and workload allocation.

Several methods for measuring skill levels exist. Their main ingredient is information about the abilities and experience of a worker. This information can be gathered in various ways, e.g., by observing the worker, by sample works and tests, by querying the supervisor of the worker, or by self-reporting in an interview or through a questionnaire (cf. Gnahs, 2010, pp. 48–58; Schaper, 2007). Methods to measure skill levels often combine different of these ways to gather information about the efficiency with which a worker masters a skill. A specific method of measurement should meet quality standards with respect to objectivity, reliability, and validity. Additionally, a method should be economical, ethical, and accepted by the people whose skills are evaluated (cf. Gnahs, 2010, pp. 52–54; Erpenbeck and Rosenstiel, 2007, p. XXVIII).

To ensure a systematic storage, regular updates, and controlled accessibility of skill level data, a skill management system is needful. An important foundation of a skill management system is a database that records the skills and skill levels of workers. Responsible for regular updates of skill data are supervisors, department managers, members of the human resource department, or workers themselves. Managers that have to make staffing decisions can use the database to find suitable workers for tasks and algorithms that support these decisions can be linked to the database. Other uses of the data include decisions about training of workers, for instance. Examples for skill management systems used in practice are given by Beck (2003), Hiermann and Höfferer (2005), and Pawlowsky et al. (2005, pp. 360–362). A skill management system is the vital basis for an efficient use and development of a multi-skilled workforce.

Multi-skilling, which is also termed cross-training and means equipping a worker with more than one skill, has many advantages for both the worker and the employer. Though, disadvantages and pitfalls exist that should be taken into account. Slomp and Molleman (2002, pp. 1194–1196) and Inman et al. (2005, pp. 117–118) discuss various impacts of cross-training. In the following we summarize their arguments for and against multi-skilling.

From an employer's perspective, a main advantage of multi-skilling is the increase in flexibility that allows to better cope with changes in skill demand and skill supply. A change or shift in skill demand can occur when customer requests rise for those products or services that heavily rely on a certain skill. Supply shifts can be caused by illness or absenteeism, for example. Increased flexibility can avoid lost sales that would arise due to a shortage in labor supply and it can save money as it can reduce the need for hiring external temporary staff when service or output levels have been guaranteed. Since external workers are less familiar with procedures and processes in the hiring firm, quality issues may emerge when external workers are engaged. On the other hand, training of internal workers that is necessary to reach a desired degree of flexibility costs money.

From an employee's point of view, multi-skilling enables job rotation which can raise motivation and satisfaction as it makes work less repetitive and provides a better overview of the work process the employee is involved in. Though, cross-training can also attenuate motivation. When the skill sets of workers assimilate, a single worker looses his expert

status and his contributions to the work team are no longer indispensable. Social loafing may occur if no one wants to undertake an unpopular, troublesome task because everyone else has the required skill to accomplish the task. Moreover, workers may oppose change and may be reluctant to learn and apply new skills. A positive impact of cross-training for the worker is that it makes him more attractive to other employers and is likely to increase his wage, what is a drawback for the current employer.

Workload sharing is another important aspect of multi-skilling and has advantages and disadvantages for employees. Cross-training enables workers to support one another and facilitates sharing of workload. This allows better workload leveling and can thus help to satisfy the desire for a just distribution of workload. At the same time, sharing of workload can reduce processing times of orders and idle times of workers. On the flip side of the coin, workers may suffer from work intensification (cf. Adams et al., 2000; Green, 2004).

A further aspect of multi-skilling is the maintenance of skills. Skills must be maintained by strategies such as job rotation or task variety but these strategies have disadvantages. If workers acquire more and more skills, they are at risk of forgetting them. Then, job rotation, job enrichment, job enlargement, or similar measures are necessary to counteract forgetting. All these measures, however, come along with problems such as greater need for coordination, higher setup costs due to frequently changing worker assignments, and tendentially lower continuity in job and task outcomes.

We conclude from the discussion, as Slomp and Molleman (2002, p. 1196) and Inman et al. (2005, p. 118) do, that cross-training is in general beneficial for both workers and employers if multi-skilling is carefully managed and its extent is limited. So, it may be wise to equip a worker with two skills and not with one or five skills, for example. But is a rather small number of skills per worker sufficient to effectively cope with fluctuations in the demand of skills? And if so, which skills should be mastered by which worker, i.e., how should a firm's skill configuration look like? And how can a given skill configuration be evaluated and compared to another one? Answers to these questions will be outlined in the final part of this section, which has a focus on the principle of skill chaining.

In the third and last part of this section, we will first consider the question how a workforce with a small number of skills per worker compares to a fully flexible workforce where each worker masters every skill. Several studies have found that a limited number of skills per worker is sufficient to yield almost all benefits of a fully flexible workforce. We will review some of these studies and realize that it is crucial how skills are distributed among workers.[4] The question of an optimal skill distribution is addressed by the flexibility design problem. A prominent but not necessarily optimal solution of this problem is the principle of skill chaining, which will be explained at the beginning of this part. At the end of this part, we provide an overview of measures that allow to quickly assess the quality of skill configurations.

In a seminal paper, Jordan and Graves (1995) examined the flexibility of manufacturing plants and outlined the chaining concept. The authors consider fictitious plants that differ with respect to their product portfolio. A plant in their paper corresponds to a worker in this thesis and the product portfolio of a plant corresponds to the set of skills

[4] All studies that we review, except those of Jordan and Graves (1995) and Campbell (1999), are classified either in Table 2.2 on page 13 or in Table 2.3 on page 15. The study of Campbell (1999) is a precursor of the work of Campbell and Diaby (2002), which is listed in Table 2.3.

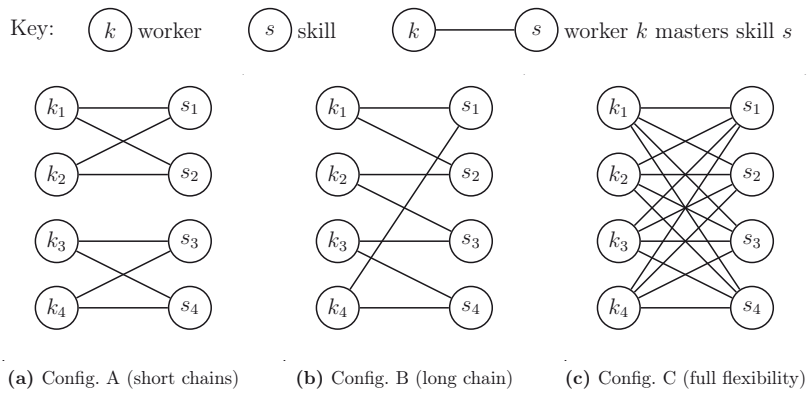
mastered by a worker. A plant where only one specific product can be manufactured is said to have no flexibility. Each additional product that can be manufactured in this plant enlarges the product portfolio of this plant and makes the plant more flexible. Given a limited portfolio size, Jordan and Graves wanted to determine an ideal product portfolio for each plant such that the expected total shortfall for a random demand for all products is minimized. They showed that for a set of plants flexibility does not only depend on the flexibility of each single plant, but also on "the opportunities ... for shifting capacity from building products with lower than expected demand to those with higher than expected demand" (Jordan and Graves, 1995, p. 580).

Shifting capacity requires direct links between plants and the capability of shifting capacity is enhanced by indirect links between plants. Two plants are directly linked if they can produce at least one common product. Two plants are indirectly linked if a chain of direct links connects both plants (cf. Jordan and Graves, 1995, pp. 579–582). The longer the chains are, the greater the opportunity for shifting capacity and, hence, the greater the flexibility of the set of plants. Analogously, in our case two workers are directly linked if they master at least one common skill and two workers are indirectly linked if they are connected by a chain of direct links. Jordan and Graves compared configurations with short and long chains and demonstrated the advantage of long chains, especially the advantage of a closed chain of maximum length. If such a chain links all plants, limited flexibility at the plant level is sufficient to achieve almost the same benefits as in the case of full flexibility at the plant level. Therefore, Jordan and Graves recommended the concept of chaining, i.e., designing configurations with closed, long chains.

Before we continue the literature review, we will give an example for different skill configurations and illustrate the concept of chaining. Assume that a set $\mathcal{K} = \{1, \ldots, K\}$ of workers or resources is given where an element of $\mathcal{K}$ is denoted by k. Let $\mathcal{S} = \{1, \ldots, S\}$ denote a set of skills and let s represent a single skill from $\mathcal{S}$. For the sake of brevity, we refer to worker $k = i$ by k_i, $i = 1, \ldots, K$, and we abbreviate skill $s = i$ by s_i, $i = 1, \ldots, S$. For a case of four workers and four skills, Figure 2.2 illustrates three different skill configurations. A configuration is depicted as a bipartite graph $G = (N, E)$ with node set N and edge set E. The node set comprises $K+S$ nodes. Each worker $k \in \mathcal{K}$ is associated with a node in G. The K nodes of graph G that are associated with workers are termed *worker nodes*. Likewise, each skill $s \in \mathcal{S}$ is associated with a node in G. The S nodes that correspond to skills are termed *skill nodes*. The fact that node set N is composed of worker and skill nodes is expressed by the notation $N = \mathcal{K} \cup \mathcal{S}$. The set of edges E contains only pairs of worker and skill nodes, i.e., $E \subseteq \mathcal{K} \times \mathcal{S}$. An edge $(k, s) \in E$ indicates that worker k masters skill s.

Configurations A and B in Figures 2.2(a) and 2.2(b), respectively, represent cases of limited flexibility. Note that in both configurations each worker masters two skills and each skill is mastered by two workers. Configuration A features two short chains. Configuration B features a closed chain of maximum length; this configuration is a classical *2-chain*. Configuration C represents the case of full flexibility where each worker masters every skill.

In all three configurations, workers k_1 and k_2, for example, are directly linked, because both workers master a common skill. Assume that worker k_i accomplishes the workload of skill s_i, $i = 1, \ldots, 4$. Let there be a low demand for skill s_1 and a high demand for skill s_2 which exceeds the capacity of worker k_2. Then, worker k_1 can support worker k_2



(a) Config. A (short chains) **(b)** Config. B (long chain) **(c)** Config. C (full flexibility)

Figure 2.2: Examples for skill configurations

in satisfying the demand for skill s_2. Now, assume that there is a low demand for skill s_1 and a high demand for skill s_3. In configuration B, workers k_1 and k_3 are indirectly linked, as there is a chain that connects k_1 and k_3, e.g., the chain via nodes s_1–k_1–s_2–k_2–s_3–k_3. Unused capacity of worker k_1 can be shifted along this chain to worker k_3 who faces additional demand for skill s_3. The shift of capacity works as follows. Worker k_1 satisfies demand for skill s_2, thereby freeing up capacity of worker k_2, who in turn can support worker k_3 in accomplishing workload of skill s_3. In configuration A, in constrast, workers k_1 and k_3 are not linked at all. Hence, configuration A cannot accommodate the assumed shift in demand from skill s_1 to skill s_3.

Brusco and Johns (1998) investigated different cross-training strategies for maintenance workers of a paper mill and examined the chaining concept among other things. The authors consider a model that aims at minimizing costs for assigning workers with different skill sets to skill requirements. Workers master their skills at different levels, i.e., with different efficiencies. For each skill, one requirement exists that must be staffed with an arbitrary number of workers such that the sum of their efficiencies exceeds a given threshold. Given this model, Brusco and Johns assessed various cross-training strategies that combine three basic decisions. They assume that workers master a primary skill with an efficiency of 100 % and can be cross-trained (1) for one or two secondary skills (2) with an efficiency of either 50 % or 100 % (3) such that either short or long chains as defined by Jordan and Graves (1995) are created.

Brusco and Johns found that a limited amount of flexibility, i.e., one secondary skill with an efficiency of 50 %, leads to substantially better solutions compared to the situation without cross-training and yields almost as good solutions as two secondary skills with efficiencies of 100 %. Furthermore, cross-training strategies that lead to long skill chains facilitate considerably greater cost savings than strategies that lead to short chains.

Campbell (1999) examined different cross-training policies for the problem of assigning nurses to departments of a hospital at the beginning of a shift. Nurses can be qualified to work for other departments than their primary department. The efficiency with which

a nurse works for her secondary departments can vary from department to department and from nurse to nurse. Campbell formulated a model that minimizes the total shortfall of nurses across all departments. The shortfall of a department does not only depend on the number of nurses assigned to this department, but also on their efficiencies. In his experiments, Campbell varied the cross-training breadth and the cross-training depth. The breadth of cross-training is determined by the average number of departments for which a nurse can work, the depth is defined by the minimum efficiency for secondary departments.

Also Campbell has found that limited flexibility is sufficient to yield virtually as high benefits as in the case of total flexibility. In other words, the study revealed that marginal returns of multi-skilling decrease. Furthermore, the results of Campbell indicate that a low level of cross-training depth can be compensated by greater cross-training breadth and vice versa.

Gomar et al. (2002) modeled a staffing problem that arises for construction projects. They examined the effect of multi-skilling on two measures. The first measure is the ratio of hires to the maximum number of required workers in the course of the project. This ratio is expected to decrease with increasing multi-skilling, because a multi-skilled worker may be assigned to another task when a task has been finished, whereas a mono-skilled worker who is not skilled for the next task is fired and a new worker is hired. The second measure is the mean tenure of workers during a construction project.

Gomar et al. found that multi-skilling has a positive impact on both measures. The higher the share of multi-skilled workers in the workforce, the smaller is the ratio of hires to required workers. The more skills mastered by a worker, the longer is his tenure on average. Notably, the growth of both benefits diminishes when the degree of multi-skilling increases. Thus, the results of Gomar et al. are in line with the results of previously cited studies.

Another study that investigated the value of cross-training was presented by Vairaktarakis (2003). He formulated a model for a project scheduling problem where each activity must be processed by exactly one qualified resource. Each resource can process only a subset of all activities and resource efficiencies are either 0 or 1. The model pursues the minimization of project duration. Vairaktarakis varied the capabilities of the resources and recorded the deviation of the resulting project duration from the duration that is achieved in case of full flexibility. We term the latter project duration *ideal* project duration. The author investigated the range from inflexible resources that can process only a small subset of all tasks to fully flexible resources that can process any task. At around half the way from inflexibility to full flexibility, deviation from the ideal project duration was down from about 30 % to less than 5 % for almost all test sets.

Recently, Heimerl and Kolisch (2010a) analyzed the impact of multi-skilling among other things in a model that minimizes costs for staffing IT projects. For these projects, a start time must be determined and workload must be allocated to multi-skilled workers who master different skills at different levels. Heimerl and Kolisch take only variable costs into account that depend on the time that a worker performs a skill. These variable costs vary from worker to worker. When the number of skills mastered by a worker increases, staffing costs decrease, because some workers master their additional skill at a high level. These workers can substitute less skilled workers. Again, savings generated by

an additional skill per worker decrease when the number of skills mastered by a worker increases.

In conclusion, the studies found in the literature agree on two results. First, with limited flexibility one can yield almost as high benefits as with total flexibility. In other words, marginal benefits of flexibility tend to decrease. Second, chaining is advantageous in general. However, chaining may not be optimal in every case, e.g., when expected skill demands differ widely across skills while capacities are similar across resources, and sometimes chaining may not be possible, e.g., when budget constraints limit the extent of cross-training. Then, other skill configurations can be optimal. The so called *flexibility design problem* addresses the task to find an optimal skill configuration.

For a specific variant of the flexibility design problem, Schneider et al. (2013) provide a mixed-integer programming formulation. The variant is formulated as a two-stage stochastic program with fixed and complete recourse (cf. Birge and Louveaux, 1997, pp. 84–93 and 163–165; Shapiro et al., 2009, p. 33). On the first stage, decisions about the skill configuration must be made, while skill demands are unknown but all possible scenarios for demand realizations are given and the probability of occurrence is known for each scenario. Costs are incurred for equipping resources with skills. On the second stage where uncertainty about skill demand is resolved, an optimal contribution of each resource to each skill considering the first-stage decisions can be determined. Revenues are earned for each unit of satisfied skill demand.

The variant of the flexibility design problem tackled by Schneider et al. is strongly NP-hard, as can be shown by transformation from the fixed-charge network flow problem, which is also known as the minimum edge-cost flow problem (cf. Garey and Johnson, 1979, p. 214; see also page 83 in Subsection 4.6.3 of this thesis). To solve larger-sized instances of their problem heuristically, Schneider et al. devise a genetic algorithm.

The approach of Schneider et al. draws on detailed information about distribution of skill demand. For situations where less information is available, Iravani et al. (2005) and Chou et al. (2011) have outlined simple but meaningful flexibility measures that can be used to assess skill configurations and to design good configurations. For the following explanation of their measures, we assume that a skill configuration with K workers and S skills is given and that the configuration is represented by a bipartite graph $G = (N, E)$ with $N = \mathcal{K} \cup \mathcal{S}$. As already mentioned, the two sets of nodes that constitute node set N are termed worker nodes and skill nodes, respectively.

Two out of the three flexibility measures proposed by Iravani et al. (2005) are based on the so called *structure flexibility matrix* M, which is a quadratic, symmetric matrix with S rows and S columns. An entry $m_{ss'}$ of M where $s, s' \in \mathcal{S}$ and $s \neq s'$ states the number of non-overlapping paths between skill node s and skill node s' in graph G. The number of non-overlapping paths, in turn, is equal to the number of opportunities to shift capacity that is not required for skill s' to skill s. The value of such an entry $m_{ss'}$ can be calculated by solving a maximum flow problem. The entries m_{ss}, $s \in \mathcal{S}$, which lie on the main diagonal of matrix M, represent the number of edges incident with skill node s, i.e., the number of workers that master skill s. The structure flexibility matrices of the three skill configurations A, B, and C depicted in Figure 2.2 look as follows.

$$M_A = \begin{pmatrix} 2 & 2 & 0 & 0 \\ 2 & 2 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \end{pmatrix} \qquad M_B = \begin{pmatrix} 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{pmatrix} \qquad M_C = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \end{pmatrix}$$

Iravani et al. propose the three measures φ_{arc}, $\varphi_{\text{mean}}(M)$, and $\varphi_{\text{eigenvalue}}(M)$ to indicate the flexibility of a skill configuration with structure flexibility matrix M. Computation of the latter two is based on matrix M, whereas φ_{arc} can be determined without knowing M. The measure φ_{arc} is the most simple one. It is defined as the total number of edges $|E|$ in bipartite graph G that represents the skill configuration; an equivalent definition is given by $\varphi_{\text{arc}} := \sum_{s \in \mathcal{S}} m_{ss}$. The measure $\varphi_{\text{mean}}(M)$ represents the mean value of all elements in M; in mathematical terms, it is defined as $\varphi_{\text{mean}}(M) := \frac{1}{S^2} \sum_{s \in \mathcal{S}} \sum_{s' \in \mathcal{S}} m_{ss'}$. The measure $\varphi_{\text{eigenvalue}}(M)$ is defined as the largest eigenvalue of the matrix M, which has S eigenvalues. Iravani et al. recommend to apply the measure $\varphi_{\text{eigenvalue}}(M)$ only to connected graphs G, whose structure flexibility matrix has only strictly positive entries $m_{ss'}$, s, $s' \in \mathcal{S}$. For each of the three measures, it is understood that the flexibility of a skill configuration is the greater, the larger the value of the measure is.

Chou et al. (2011) have presented a set of flexibility measures that require more information; the authors presume that information about resource capacity and expected skill demands is available. More precise, they assume that for each worker or resource $k \in \mathcal{K}$, its capacity c_k and for each skill $s \in \mathcal{S}$ its expected demand $E[\tilde{d}_s]$ is known. Here, $\tilde{d}_s$ is a random variable that represents the uncertain demand for skill s and $E[\,\cdot\,]$ is the expected value operator.

As flexibility measures Chou et al. calculate so called *expansion ratios* for various subsets of nodes. These measures have been derived from desired characteristics of flexible skill configurations. The two most meaningful expansion ratios are the ratio δ_i^{node}, which is associated with a single node $i \in N$ of configuration graph $G = (N, E)$, and the ratio $\delta_{ii'}^{\text{pairwise}}$, $i < i'$, which is associated with nodes i and i' that both belong either to the subset $\mathcal{K}$ or to the subset $\mathcal{S}$ of the node set N. For each of these two expansion ratios, the definition differs between worker nodes and skill nodes.

The single node expansion ratios are defined as follows (cf. Chou et al., 2011, p. 1100).

$$\delta_k^{\text{node}} := \frac{\sum\limits_{s \in \mathcal{S} \,|\, (k,s) \in E} E[\tilde{d}_s]}{c_k} \qquad k \in \mathcal{K}, \qquad\qquad \delta_s^{\text{node}} := \frac{\sum\limits_{k \in \mathcal{K} \,|\, (k,s) \in E} c_k}{E[\tilde{d}_s]} \qquad s \in \mathcal{S}$$

The larger the smallest δ_i^{node}, $i \in \mathcal{K} \cup \mathcal{S}$, for a skill configuration, the greater is its flexibility. A large δ_s^{node}, $s \in \mathcal{S}$, indicates that the demand for skill s is likely to be satisfied due to sufficient capacity. A large δ_k^{node}, $k \in \mathcal{K}$, signals that the capacity of resource k is likely to be used and unlikely to be idle. If the smallest single node expansion ratios of two skill configurations are equal, ties are broken by comparing the next lowest expansion ratios.

The definitions of the pairwise expansion ratios read as follows.

$$\delta_{kk'}^{\text{pairwise}} := \frac{\sum\limits_{s \in \mathcal{S} \,|\, (k,s) \in E \vee (k',s) \in E} E[\tilde{d}_s]}{c_k + c_{k'}} \qquad\qquad (k, k') \in \mathcal{K} \times \mathcal{K},\ k < k'$$

$$\delta^{\text{pairwise}}_{ss'} := \frac{\sum_{k\in\mathcal{K}\,|\,(k,s)\in E\,\vee\,(k,s')\in E} c_k}{E\big[\tilde{d}_s\big] + E\big[\tilde{d}_{s'}\big]} \qquad (s,s') \in \mathcal{S}\times\mathcal{S},\ s<s'$$

Again, the greater the smallest $\delta^{\text{pairwise}}_{ii'}$, $(i,i') \in \{\mathcal{K}\times\mathcal{K} \cup \mathcal{S}\times\mathcal{S}\}$, $i<i'$, the more flexible is a skill configuration. Ties are broken by comparing the second smallest pairwise expansion ratios.

To give an example, we applied the flexibility measures of Iravani et al. (2005) and Chou et al. (2011) to the three skill configurations depicted in Figure 2.2. For all configurations, we assumed $c_k = 1$ for each resource $k \in \mathcal{K}$ and $E\big[\tilde{d}_s\big] = 1$ for each skill $s \in \mathcal{S}$. The values of the measures are displayed in Table 2.4. Note that the measures differ in their discriminatory power. Only the measures φ_{mean}, $\varphi_{\text{eigenvalue}}$, and $\delta^{\text{pairwise}}_{ii'}$ clearly rank the three configurations and indicate that skill configuration C is the most flexible and configuration A the least flexible. For configurations A and B, all single node expansion ratios are of identical value. Ties are broken by the lowest pairwise expansion ratio, which marks configuration B, which follows the classical chaining principle, as more flexible than configuration A. However, when the number of skills per worker is limited, classical chaining need not lead to the best possible configuration in general, as Chou et al. (2011, pp. 1100–1102) have shown by comparing a 3-chain to a so called Levi graph.

Table 2.4: Values of different flexibility measures for the three skill configurations from Figure 2.2 (a bracketed value for $\varphi_{\text{eigenvalue}}$ indicates that this value was computed—contrary to recommendation—for a disconnected graph)

	Skill configurations from Figure 2.2		
Flexibility measure	Config. A (short chains)	Config. B (long chain)	Config. C (full flexibility)
φ_{arc}	8	8	16
φ_{mean}	1	2	4
$\varphi_{\text{eigenvalue}}$	(4)	8	16
Lowest δ^{node}_i	2	2	4
Lowest $\delta^{\text{pairwise}}_{ii'}$	1	1.5	2

A skill configuration that allows to effectively handle demand shifts is advantageous when it comes to project selection. The more flexible the skill configuration, the larger is the set of feasible portfolios, because each portfolio represents a scenario for skill demand. In Section 7.2, we will present computational results that confirm this statement. In regard to the workforce assignment problem, a skill configuration with a long chain is also advantageous compared to a configuration with short chains for forming small teams, as preliminary tests revealed. But before the formation of small teams will be addressed, we want to explain in the first place why small teams are advantageous.

2.3 Why small project teams?

The objective of our main problem, the problem of assigning workers to projects, is to minimize average project team size. In this section, we motivate and justify this objective;

we argue that project teams which have a small number of members in relation to project workload are advantageous compared to relatively large teams. Before we address the question "Why small teams?", we respond to the question "Why teams, why teamwork?" We define the terms *team* and *project team*, outline advantages of working in groups compared to working individually, and point to some important issues such as social competencies and team size that should be taken into account when managing work teams. However, teamwork has not only advantages; teamwork also poses problems. A problem relevant in the context of this thesis is social loafing, i.e., the phenomenon that workers tend to expend less effort when working in teams than when working alone. We report on experimental and empirical findings that have demonstrated the existence of this phenomenon and identified moderating variables. Among these variables is team size, which is related positively to social loafing: Productivity losses of team members are the greater, the larger a team is. A small team has further advantages and we describe benefits with respect to communication and division of labor, for example. We provide experimental, empirical, and theoretical evidence from literature for the superior performance of relatively small work groups in comparison to large groups. Finally, we consider advantages of relatively small project teams from an employee's perspective who works in a multi-project environment.

Following Antoni and Bungard (2004, p. 138), we define a *team* as a group of persons who have a common task and must cooperate to accomplish their task. Ideally, team members share common goals and perceive themselves as a team. We understand a *project team* as a temporary team that has to solve a new, complex problem and that may consist of persons who are experts in different fields (cf. Antoni and Bungard, 2004, p. 140; Mintzberg, 1980, pp. 336–337).

Collective work, which means the organization of work in teams or groups[5] and which is frequently encountered in almost every organization, has several advantages in comparison with individual work. First of all, there are tasks that cannot be accomplished by an individual but require the joint effort of several persons, e.g., the task of building a bridge, the task of lifting a heavy table and carrying it from the basement to the second floor, the task of redesigning the process of handling an insurance claim, or the task of designing a new car model. Furthermore, synergies between group members who contribute expertise from different areas can lead to innovative solutions. Teamwork can have a positive impact on motivation and job satisfaction of workers because workers who are part of a team may experience mutual encouragement and assistance, realize the importance of their contribution for the success of the group, and receive credit and acknowledgment from fellow team members (cf. Hertel, 2011, p. 176). Positive impacts exist especially when a certain degree of discretion is granted to the team allowing the team to make own decisions.

Teamwork has benefits not only for workers but also for the organization as a whole. Delarue et al. (2008) reviewed 31 studies in which employees or managers were surveyed or company documents were analyzed. They found support for the hypothesis that teamwork is related positively to organizational performance. Delarue et al. could confirm that teamwork lowered throughput times, increased job satisfaction, and reduced absenteeism; these effects in turn improved productivity and profitability.

Examples for surveys that have been included in the meta-analysis of Delarue et al.

[5] We use the terms *team* and *group* interchangeably.

(2008) are the surveys of Banker et al. (1996) and Elmuti (1997). Banker et al. (1996) conducted a longitudinal study of a U.S. manufacturing plant where electromotors are produced. The study spanned almost two years and covered the introduction of work teams. Banker et al. screened company records to obtain data on quality, measured as the defect rate of produced units, and on labor productivity, measured as output per production hour. The authors found that quality and labor productivity improved significantly after the implementation of work teams.

Elmuti (1997) conducted a cross-sectional study of U.S. firms to evaluate the impact of self-managed teams on organizational performance. 126 managers from firms in various sectors returned the questionnaire developed by Elmuti. His analysis of the answers confirmed a positive link between teamwork and organizational performance. Organizational performance was measured as a mix of productivity, product quality, service quality, and other performance indicators.

Teamwork is a complex process, which has not been fully understood yet. Different theories about group processes have been proposed that exhibit similarities but have not been fully aligned so far. Hence, researchers have not agreed on clear guidelines for managing teams. Two theories of team processes shall be mentioned. Both, the theory of Tuckman and Jensen (1977) and that of McGrath (1991), state that when team members have been selected and their project has been started, teams usually do not work effectively right away and perform their task immediately.

From a literature review, Tuckman and Jensen (1977) derived a five-stage model that describes the development of a group. Before a group actually performs (stage 4, *performing*), there is a phase in which members get to know each other and their common task (stage 1, *forming*), followed by a phase in which conflicts about roles and positions within the group break forth and competing solutions for the team task are brought forward (stage 2, *storming*). To resolve these issues, norms and rules for the collaboration must be established (stage 3, *norming*). After the workload has been accomplished, the collaboration ends and the group may break up (stage 5, *adjourning*).

According to the theory of McGrath (1991), there is no fixed sequence of stages a group runs through. Instead there are rather four alternative modes of activities that group members can exert at any time. These modes are titled *inception*, *problem solving*, *conflict resolution*, and *execution*. McGrath argues that engagement in conflict resolution is not a sign of team failure but reflects difficult conditions under which a team has to search for a viable path for attaining its goals.

In our models, which deal with the work of project teams in a firm, we will focus on the performing stage or on the execution mode, respectively, and neglect the remaining aspects. We assume that performing or executing, i.e., accomplishing workload, occupies most of the time that is required by a project team to complete its task. Nevertheless, it should be kept in mind that some time will be taken by those aspects that we ignore.

It is widely accepted that the design and the composition of a team are crucial for effective teamwork. Important design criteria stated by Högl (1998, pp. 88–108) are competencies of team members, heterogeneity, leadership, and team size. Högl (1998, pp. 89–91 and 149–159) and Klimoski and Jones (1995, pp. 311–312) stress the importance of social competences as a key difference to the requirements of individual work.

With regard to heterogeneity, Zenger and Lawrence (1989) surveyed the members of 19 project teams in a U.S. electronics firm. They found that team members of similar

age communicate more often about their project tasks than members who are dissimilar in terms of age. Communication across team boundaries is more likely to occur between employees who are similar to each other in terms of tenure. Although communication influences team outcomes positively, Zenger and Lawrence (1989, pp. 372–373) warn against staffing each team in a firm with employees of similar age because these team compositions may have negative implications in the long run. In regard to leadership, an overview of leadership theories is presented by Wilke and Meertens (1994, pp. 156–188), who point out that an explicit group leader is not absolutely necessary and only beneficial if his net effect with respect to coordination and motivation is positive.

Team size, which is defined as the number of team members, is a design variable that has occupied researchers for a long time. Already Simmel (1902, pp. 159–160) noted that an additional group member changes the relationships within a group and enhances deindividualization; this effect of an additional member is the greater, the smaller the group is. Simmel (1902, pp. 192–196) gives examples from various epochs and cultures where the size of certain groups was constrained. He argues that an upper limit on group size was set to avoid that a member perceives her individual responsibility as negligible. The purpose of a lower limit on group size, e.g., for groups with decisive power, was to facilitate corrective actions in discussions of the group and to attenuate the influence of members with extreme positions or opinions.

In an early literature review, Thomas and Fink (1963) analyzed 31 empirical studies that examined effects of group size. Thomas and Fink differentiated between effects on the group as a whole and effects on group members. For the group as a whole, they concluded that increasing team size leads to a decrease in cohesion and to the emergence of subgroups. For individual members, satisfaction tends to decrease as group size increases. However, the authors point to methodological shortcomings of the studies, such as uncontrolled independent variables other than team size that may affect dependent variables.

In general, the appropriate team size depends on the nature of the team task and on its requirements. If the nature of the task is associated with creativity, e.g., when ideas for a new advertising campaign shall be generated, the adequate team size tends to be larger than in a case where a team has to make many decisions in a short period of time and where consensus is necessary, e.g., when a military operation is executed. If the nature of the task stays the same but requirements change, optimal group size can change. As an example, consider the required speed for decision making. When speed becomes a critical issue, smaller groups of decision makers are often favored, whereas larger groups are preferred when more time is available, as can be seen from many constitutions that prescribe different rules for legislation in wartime and in times of peace.

From the team task or workload, a lower and an upper limit on team size can be derived. Klimoski and Jones (1995, pp. 307–308) emphasize that a group must be so large that the capacity of the members is sufficient to accomplish the task. With respect to a reasonable maximum size of a team, researchers tend to recommend that a team should have only as many members as necessary. Filley et al. (1976, pp. 138, 144–147, and 417–421) conclude from various studies that groups should be rather small and Hackman (1987, p. 327) states that a team should be “just large enough to do the work”. Hence, the lower limit and the upper limit on team size coincide and researchers advise the

smallest possible team size as the appropriate size. Reasons for this recommendation are disadvantages of teamwork that intensify with increasing team size.

One such disadvantage of teamwork is *social loafing*, which names the phenomenon that individuals tend to exert less effort when working in groups than when working alone. This dysfunctional effect of groups was first observed for physical tasks in laboratory experiments. Ingham et al. (1974) repeated the experiment of the French agricultural engineer Maximilien Ringelmann where participants had to pull a rope with maximum strength alone and in groups. Measurements showed that average individual effort decreased with group size. Additionally, Ingham et al. measured the performance of participants when pulling in pseudo-groups of various size. Here, participants were made to believe that they pulled in a group but actually pulled alone. The pseudo-group experiments have shown that the decline in effort can be explained only partially by coordination losses and must be attributed mainly to a loss in motivation. The loafing effect was also observed by Latané et al. (1979) in experiments where subjects had to clap and shout as loud as possible. In these experiments, the size of the audience was kept constant to control motivational effects that arise from evaluation apprehension.

Social loafing can have different causes. Several theories try to explain the loafing phenomenon and each theory emphasizes one cause or another (cf. Karau and Williams, 1993, pp. 682–684). According to social impact theory, team members are targets of social impact. The social impact can emanate from a supervisor who assigns a task, for instance. This impact is divided equally across the targets. Hence, an increase in team size reduces the impact on each team member and leads to a decline in individual effort. Note that a team member need not be fully aware of his reduction in effort as it is likely for the rope pulling experiment, for example.

Another explanation of social loafing refers to evaluation potential. If the effort or input of a group member cannot be observed and thus cannot be evaluated, this member may exert only little effort as he or she cannot be blamed if the team misses its goals. Under these circumstances, "the opportunity to 'hide in the crowd' or otherwise gain respite from the pressure of constant scrutiny" is attractive and this opportunity occurs the more often, the larger the group is (Davis, 1969, p. 72). But even if it is likely that goals will be achieved, an individual may slack if he fears that his contribution cannot be identified and that he will not receive appropriate rewards or an adequate appraisal. This motivation loss is ascribed to feeling "lost in the crowd" (Latané et al., 1979, p. 830); this loss also grows with group size.

Social loafing of team members can be a consequence of rational behavior, which is in accordance with self-interest, but can also be based on irrational behavior, which can be explained by social values, as Kerr (1983) demonstrated. His experiments revealed two effects that can come along with group work, namely, the free-rider effect, which can be attributed to rational behavior, and the so called sucker effect, which cannot be explained with rational, self-benefiting behavior. Loafing of a group member is called free riding when the member realizes that his effort is dispensable and that he can profit from group outcome without making a real contribution and then willfully expends only little effort. The sucker effect describes a reduction in effort that is a response to free riding of fellow team members. The effect occurs when a group member free rides and another member reduces his effort because he feels being exploited by the free rider and does not want to play the sucker role. Kerr observed this retributive inactivity although it was irrational

in his experiment because it was associated with reduced payoffs. He inferred that this behavior was driven by equity norms. Both, free riding and retributive inactivity, are forms of social loafing.

Free riding is more likely to occur in larger than in smaller groups. Hamburger et al. (1975) provided experimental evidence for this claim. They let groups of three and seven persons play 150 rounds of a take-some game where payoff structures were comparable for both group sizes. In each round of the game, a player can choose one of two strategies: either cooperation or defection. Overall expected returns in a round are highest when all players cooperate and lowest when all players defect. Though, a player can always increase his individual expected returns by choosing to defect. Such an incentive structure is typical for so called social dilemmas. In the experiment, defection rates were higher for the groups of seven. Hamburger et al. concluded that the decline in cooperation for the larger groups was caused by deindividuation[6], i.e., by greater anonymity implying a weaker feeling of individual responsibility. Hence, free riding and, consequently, the sucker effect occur with greater probability in large groups than in small ones.

Bonatti and Hörner (2011) modeled free riding in a theoretic approach. They consider individuals who collaborate in a project. Project success is uncertain and depends on the total effort expended by team members over time. Each individual decides about his effort level at every point in time. The model suggests that individuals will engage in free riding and postpone their contributions to later points in time, no matter if team members can or cannot observe the effort choice of fellow team members. In fact, the model predicts that procrastination of the project will be the greater, the larger the project team is.

It is worth noting that social loafing is a phenomenon that prevails in individualistic cultures but is rather rare in collectivistic, group-oriented cultures (cf. Gabrenya et al., 1985; Earley, 1989, 1993). Earley (1993), for instance, compared the performance of American, Chinese, and Israeli managers who had to accomplish an in-basket task. He examined three work situations: working alone; working in a pseudo-ingroup that allegedly comprised subjects with similar personality and similar other characteristics such as religious orientation and lifestyle; and working in a pseudo-outgroup whose members were said to have not much in common as they allegedly had different interests, backgrounds, and lifestyles. American participants, whose cultural background is understood as individualistic, performed best when working alone. Chinese and Israeli managers, who live in societies characterized by collectivism, performed better in ingroups than in the outgroup and in the alone condition.

Several variables besides culture moderate the extent of social loafing. Overviews of moderating variables are contained in the articles of Karau and Williams (1993), Comer (1995), and Oelsnitz and Busch (2006). Social loafing can be reduced by a motivating task or intrinsic motivation; by task visibility, i.e., evaluation potential; by perceived indispensability of one's contribution; by perceived influence over the group outcome; and by a small group size. Note that a small group size has positive effects on many of the other aforementioned variables: The smaller a group is, the greater tends to be the perceived and actual impact of a group member on the outcome of group work, for example.

Social loafing has mainly been observed in laboratory settings with ad hoc groups performing simple physical or cognitive tasks. Groups were in most cases composed

[6]The terms *deindividuation* and *deindividualizaiton* have the same meaning.

of students without a common past and future. Consequently, it has been called into question whether the laboratory findings apply to real teams, e.g., work teams in firms, whose members are more familiar with one another and more likely to work together again, e.g., in a future project team (cf. Antoni and Bungard, 2004, pp. 168–169). Though, there is also empirical evidence for the loafing effect, that means field evidence from real teams.

George (1992) questioned salespeople and their supervisors of a U.S. retail chain. The salespeople worked—organized in teams—in retail stores of the chain. From the answers of 221 salespersons and 26 supervisors, George inferred that social loafing was the higher, the lower perceived task visibility and intrinsic motivation were.

Liden et al. (2004) surveyed 168 employees from 23 work groups in two U.S. manufacturing firms and interviewed 23 of their superiors. The study examined predictors of social loafing that are associated either with individuals or with the group as a whole. Among the predictors on the group level was team size and the perceived loafing of fellow team members. The authors found that team size was positively correlated with social loafing. Interestingly, Liden et al. found that perceived loafing of co-workers was negatively correlated with loafing. The latter result implies that the sucker effect did not occur or did at least not dominate and that a perceived lack of effort was compensated by others who expended more effort.

The hypothesis that group size is positively related to free riding was tested by He (2012). His study has experimental and empirical character. Over 200 students formed teams with two, three, and four members in order to do a compulsory coursework. The sample comprised 9 two-member teams, 59 three-member teams, and 21 four-member teams. Each team had to tackle a software development task which had to be completed within five weeks. All students were surveyed at the beginning and end of the five-week period. The author found a positive correlation between group size and free riding.

The finding that social loafing intensifies with increasing team size means that a worker is less productive in larger than in smaller teams. Hence, relatively large teams imply an inefficient use of human resources.

Apart from social loafing, there are other disadvantages of large teams. A serious disadvantage are greater communication requirements. Communication within teams is vital for sharing important information and coordinating member efforts. The larger a team is, the more communication links arise, independently from the way how the communication network of the team is configured.[7] In the general case where free communication among team members is possible, a larger team implies that more information must be sent as well as received and processed by each team member to keep the whole team up to date. Greater communication requirements in larger project teams can prolong project duration. If communication demands cannot be met, project quality may decrease.

Although communication requirements tend to rise with team size, the average amount of communication between two team members can actually diminish with increasing team size. Zenger and Lawrence (1989) made this observation in their empirical study of work groups in a U.S. electronics firm.[8] They found that the average frequency of commu-

[7] Davis (1969, pp. 94–104) and Wilke and Meertens (cf. 1994, pp. 183–186) depict different communication networks and discuss their impact on group performance and on the motivation of group members. In addition, Davis (1969, pp. 88–104) puts communication networks and patterns in relation to group structures.

[8] We have already mentioned this study on page 29.

nication between two team members decreased when team size increased. However, this reduction in bilateral communication can occur even if the total amount of communication increases. The decrease in communication for each pair can have several causes; it can be explained by a reduced requirement for information sharing between two team members when an additional team member decreased their individual workload or by stress and exhaustion due to the increased number of communication partners.

A lower frequency of communication between any two team members can lead to a decrease in cohesion and cooperation. The relationship between communication and cooperation was studied by Balliet (2010). He provides a meta-analysis of the literature on social dilemma experiments. The take-some game mentioned on page 32 is an example for a social dilemma, where free riding is rational for each individual but leads to the worst outcome for all participants. For these kinds of dilemmas, Balliet (2010) found that communication increases cooperation.

Tohidi and Tarokh (2006) proposed a quantitative model that links team size, communication, and productivity for a group of production workers. In their theoretical approach, workers must divide their time between production on one side and information processing including communication on the other side. For each unit of output produced by any of the workers, all workers must process one unit of information. Under the assumptions made by Tohidi and Tarokh, total output increases when team size increases but the additional output that results from a further worker decreases with team size. Since the marginal returns of team size diminish, there exists an optimal team size under certain conditions for costs, e.g., when constant costs per worker are assumed. According to their model, team size can be reduced without sacrificing output if improved information and communication technology is applied such that less time is needed for information processing.

The effect that projects take the longer, the larger the project team is, was observed by Cassera et al. (2009), for example. They analyzed data records of surgical procedures in three U.S. hospitals and found that, ceteris paribus, larger teams required more time for a surgical procedure than smaller ones. In their analysis, Cassera et al. controlled for the complexity of a procedure. A team comprised all surgeons, nurses, and other medical staff that participated in the surgical operation or observed it. Team members were present either for the complete time of the operation or for some part of it. Cassera et al. explained the efficiency loss of larger teams by greater communication requirements, which arose especially when the job of a nurse was taken over by another nurse during a procedure and the new nurse had to be informed about the current state of the procedure.

A small number of team members does not only ease communication within the team but also with stakeholders outside the team, in particular with clients in the case of external projects. The smaller the team for an external project, the smaller is the number of potential contact persons for the client and the easier it is to follow the principle of *one face to the customer*. Marketing experts recommend this principle for customer relationship management (cf. Ingram et al., 2002, p. 565). Even if the structure of a team and the configuration of the communication network within a team are geared towards a one face to the customer policy, the effort necessary to enforce this policy tends to increase with team size.

Another disadvantage of large project teams is that team composition is more likely to vary over time. Often and independent from the size of a team, some workers join the

team later in the course of the project and must be acquainted with the current state of the project. The greater a team is, the greater tends to be the number of late entrants. Each late entrant can cause an interruption when he must be trained by existing team members; moreover, he increases communication requirements also beyond the training phase. These detrimental effects of a late entrant were observed by Brooks (1982), who stated that "adding manpower to a late software project makes it later" (p. 25). Likewise, Cass (1992) concluded that varying crew levels and varying crew compositions in construction projects caused productivity losses. He drew his conclusions from an analysis of three large construction projects. For the analysis, Cass considered late entrants that replaced other team members and understaffing due to no-shows of workers.

The more workers form a team, the greater is, assuming constant workload, the division of labor. A high division of labor creates many interfaces between workers and causes many handoffs that can slow down a work process. For this reason, Hammer and Champy (1993, p. 144) recommended to reengineer processes such that as few workers as possible are involved in a process. They reported on several cases where process times could be considerably reduced by assigning a work process to a smaller number of workers (cf. Hammer and Champy, 1993, pp. 36–44, 51–53, and 182–199). In a similar vein, Brooks (1982, pp. 16–19) pointed out that it will take two workers more than half the time that it takes one worker to accomplish a task because in case of two workers additional time is necessary for communication. So, a workload of six man-months may be accomplished in six months by one worker, whereas it may take two workers three and a half month and three workers two and a half month to accomplish this workload. Brooks (1982, p. 16) warned that the man-month is not an adequate measure of task size. In our approach, we will nevertheless use this measure. However, by minimizing average project team size, we implicitly take the warning of Brooks into account.

Disadvantages of small teams shall not go unmentioned. A small project team may raise the risk of procrastination. The absence of a worker from a small project team is likely to result in a delay of the project, whereas a larger team often implies some redundancy of human resources. This redundancy allows to compensate for the absence of a worker. The lack of redundancy in small teams could be at least partially mitigated by a substitution arrangement that specifies for each member of a project team a deputy or a substitute who replaces the member in case of illness, for example. It should be ensured that each deputy is informed about project content and progress and about the task of the person he may have to replace.

Another disadvantage of small team size is related to motivation. A small team may be perceived by its members as too small and too weak to master the team task, whereas a larger team may be conceived as more likely to successfully accomplish the team workload. Hence, motivation of team members may rise and fear of team failure may decrease with increasing team size. However, in our approach we ensure that the capacities of the team members are sufficient to meet all workload requirements. And we could not find empirical support for the concern that a small but adequate team size is associated with low levels of motivation.

On the contrary, there are further empirical findings that prove the positive effect of a relatively small team size on team performance. Högl (1998) analyzed factors affecting the quality of teamwork in four software development departments in Germany that were part of three different firms. Quality of teamwork was determined by communication,

coordination, mutual support of team members, and cohesion, for example (cf. Högl, 1998, pp. 77–88). 575 managers and team members were interviewed for the analysis, which comprised 145 teams. Each team had at least three members. Högl (1998, pp. 149–151) found that absolute team size and quality of teamwork were negatively correlated, while an adequate team size, relative to the size of the task, was positively correlated with teamwork quality.

Hanna et al. (2007) assessed 103 construction projects in the United States that involved mechanical construction or metal sheet construction. For these two trades, the authors considered the impact of overmanning on efficiency, which was measured as the ratio of estimated man-hours to actually required man-hours. Overmanning was measured as absolute peak crew size and as the ratio of peak crew size to average crew size during project execution. A regression analysis showed for both measures of overmanning that efficiency decreased when overmanning increased.

Also experimental studies could demonstrate that smaller teams can perform at least as well as larger ones. As an example, we quote two studies of team performance in control rooms of nuclear power plants. In the corresponding experiments, operator teams were exposed to different scenarios in which they had to cope with critical faults. Performance was assessed by situational awareness and fault handling, for instance. For advanced control rooms, which were equipped with latest technology, Sebok (2000) found that two-person teams performed as well as four-person teams. The teams were composed of professional operators. For the experiments of Huang and Hwang (2009), students acted as operators. Huang and Hwang compared the average performance of a single operator to that of two- and three-person teams. Single operators were clearly outperformed by teams but there was no significant difference in performance between teams with two and three members.

So far we have discussed the size of a single team. When we consider a multi-project environment, which is frequently encountered in firms, there are several teams and there are further advantages of a small team size, especially for workers. In a multi-project environment, small teams imply a small average number of projects to which a worker is assigned. Small teams also imply a larger contribution of a worker per project he is assigned to. Hendriks et al. (1999) argue that small contributions to many projects are less efficient than large contributions to a few projects and hence recommend to allocate the workload of a project to a small number of workers. Inefficiencies of scattering workload across many workers and of scattering workers across many projects arise from switching and setup costs for workers and from a greater number of partners a worker has to communicate with.

Another advantage of small teams for workers is greater task variety. If a worker joins only a few projects and has to make a relatively large contribution to each project, task variety is likely to increase as he may have to accomplish different requirements in a project. Task variety fosters motivation and prevents unlearning of skills.

To sum up, there is lots of evidence that small teams are advantageous compared to relatively large ones. A small number of workers per team curbs social loafing and positively affects individual productivity. In small teams, communication is easier and cooperation and motivation are higher. For us, the advantages of relatively small teams outweigh the disadvantages and the arguments for small teams are stronger than those

for large teams. That is why we aim at small project teams in the workforce assignment problem that will be outlined in the following chapter.

Chapter 3

Problem definitions and basic notation

In this chapter, we approach our three problems, which are the selection of projects, the assignment of workers to projects, and the distribution of departmental workload. These problems are faced by an organization, e.g., a firm. The organization with its workforce and the projects that the organization can select define the setting of the three problems. Hence, we will first elaborate on the characteristics of the organization and the projects in Section 3.1 to describe the setting or the environment in which our problems are situated. For the characteristics we will introduce identifiers, which will be used through the remainder of this thesis. After having shed light on the setting, we will define the three problems, namely, the project selection problem in Section 3.2, the workforce assignment problem in Section 3.3, and the utilization leveling problem in Section 3.4.

3.1 Setting of the problems

In our problems, we consider an organization that wants to carry out projects. The link between the organization and the projects are skills that are mastered by the workers of the organization and that are required by the projects. In this section, we will first describe the firm's characteristics including skills before we turn to the characteristics of the projects. Finally, we look at skills in their role as the link between the firm's workforce and the projects.

The organization can be either a profit or a nonprofit organization, e.g., an insurance firm or a charity organization, respectively. The organization can be part of a greater organization or of a cooperation of greater organizations. For instance, the organization may be the information technology (IT) center of an insurance group. Such an IT center triggered our research. The IT center of the insurance group provides IT services for the group members. In the following, we will consider a firm as the organization.

We assume that the firm is planning its project portfolio, the assignment of workers to projects, and the allocation of departmental workload for a set $\mathcal{T} = \{1, \ldots, T\}$ of T periods, which span the planning horizon. In case of the IT center, the length of the planning horizon is one year and the length of a period $t \in \mathcal{T}$ is one month, thus $T = 12$. The top management of the IT center holds a meeting always in the last quarter of a year to plan for the the upcoming year.[1] If a project that is considered for the portfolio has a planned completion time T' later than T, the length of the planning horizon is adjusted accordingly and set to T' periods.

[1] Of course, plans are revised during the year if necessary.

M. Walter, *Multi-Project Management with a Multi-Skilled Workforce,* Produktion und Logistik, DOI 10.1007/978-3-658-08036-5_3, © Springer Fachmedien Wiesbaden 2015

The workforce of the firm is denoted by $\mathcal{K} = \{1, \ldots, K\}$, i.e., it comprises K workers. For each worker $k \in \mathcal{K}$ his availability R_{kt} is given for each period $t \in \mathcal{T}$. We measure availability in hours, hence $R_{kt} = 160$ means that worker k is available for 160 hours in period t, i.e., he can work 160 hours or less, but must not work more. The availability parameter R_{kt} allows to model full-timers, e.g., $R_{kt} = 160$, and half-timers, e.g., $R_{kt} = 80$. Furthermore, the parameter can account for a planned reduction of availability, which occurs when vacation, training, routine jobs, or other tasks are scheduled for a worker in advance.

An availability of 160 hours per month may reflect a typical labor contract that prescribes eight hours of work per day and 40 hours per week, presuming four weeks per month. If a plan assigns a workload of 160 hours to a worker whose availability is equal to 160 hours, we regard the working time limit of eight hours per day as observed, because on average eight hours per day are not exceeded. However, the actual working hours on a single day can deviate from the average when the plan is put into operation. A monthly availability limit may seem rough, but it features built-in flexibility, as the daily working times within a period can be adjusted to meet the needs of both the workers and the managers of the firm (cf. Schlereth, 2009). Furthermore, such a rough approach is adequate for a planning horizon of one year, where it is hardly reasonable to plan workloads on a daily basis several months in advance.

We assume that the workforce is fixed during the planning horizon. Short-term decisions on hires, dismissals, and resignations within the planning horizon are not taken into account. With respect to workers who are dismissed or who hand in their notices, we may likewise suppose that these workers can be immediately and adequately replaced by new hires. If hires, lay-offs or similar changes are known at the time of planning, these events can be modeled by setting $R_{kt} := 0$ for each period $t \in \mathcal{T}$ in which a worker k is not under contract. Moreover, virtually increasing the workforce by delegating work to external service providers is not considered an option.

The long-term organizational structure of the firm is reflected by a set $\mathcal{D} = \{1, \ldots, D\}$ of D departments. This structure may emphasize functions, product lines, or customer regions, for example (cf. Laux and Liermann, 2003, pp. 181–182; Schulte-Zurhausen, 1999, pp. 237–254). With regard to an IT center, there may be—amongst others— separate departments for software maintenance, database applications, and customer support. Let $\mathcal{K}_d \subseteq \mathcal{K}$, $d \in \mathcal{D}$, denote the workers that belong to department d. Since each worker k, $k \in \mathcal{K}$, is member of exactly one department, we have $\bigcup_{d \in \mathcal{D}} \mathcal{K}_d = \mathcal{K}$ and $\mathcal{K}_d \cap \mathcal{K}_{d'} = \emptyset$ for all $(d, d') \in \mathcal{D} \times \mathcal{D}$, $d \neq d'$. Let the index $d(k)$, $k \in \mathcal{K}$, denote the department to which worker k belongs.

We assume that in each department $d \in \mathcal{D}$ a departmental work requirement rd_{dt} has to be accomplished in each period $t \in \mathcal{T}$ by the staff $\mathcal{K}_d$ of department d. The departmental work requirement rd_{dt} is expressed in man-hours, so a requirement $rd_{dt} = 100$ can be accomplished, for example, by one worker $k \in \mathcal{K}_d$ who works for 100 hours or by two workers of department d who work for 50 hours each. Within a software maintenance department, the departmental work requirement can comprise the roll-out of a software update or the negotiation about a license agreement. We presume that the work requirement of a department d can be accomplished by an arbitrary subset of workers of $\mathcal{K}_d$ and that every worker of department d performs departmental workload

with the same efficiency. Thus, it takes each worker $k \in \mathcal{K}_d$ one hour to accomplish one hour of the work requirement of department d.

Competencies of the firm originate from a set of skills $\mathcal{S} = \{1, \ldots, S\}$ which are mastered by its workforce. Each worker $k \in \mathcal{K}$ masters at least one skill $s \in \mathcal{S}$. In case of the aforementioned IT center of an insurance group, skills might be experience in developing databases, experience in object-oriented programming, or knowledge of life insurance business. Let $\mathcal{S}_k \subseteq \mathcal{S}$ denote the set of skills mastered by worker $k \in \mathcal{K}$. Conversely, let $\mathcal{K}_s \subseteq \mathcal{K}$ denote the set of workers who master skill $s \in \mathcal{S}$. Sometimes there may be a close relation between skills and departments such that all the workers of a department share a common skill, which is mastered only by few workers outside of that department.

For each skill $s \in \mathcal{S}$, we distinguish four levels of efficiency with which a worker k can master skill s. The level l_{ks} lies in the domain $\{0.5, 1, 1.5, 2\}$. If worker k does not master skill s, his skill level l_{ks} is not defined. If worker k has little experience in skill s, if he is beginner or learner, his skill level l_{ks} is 0.5. This level implies that it takes worker k two hours to accomplish a nominal requirement for skill s of one man-hour. A level of 1 designates a worker who has already gained some experience in skill s. It takes him one hour to perform a requirement for skill s of one man-hour. A worker has a level of 1.5 if he is very familiar with tasks asking for skill s. He can perform a requirement of one man-hour within 40 minutes. A worker who is very well qualified and has much experience in regard to skill s is characterized by a level of 2. We presume that he needs only 30 minutes for a requirement of one man-hour. In the literature, it is common practice to distinguish four or five levels of efficiency including a level of zero, which indicates the lack of the skill (cf., e.g., Yoshimura et al., 2006, pp. 836–837; Mansfield, 1996, pp. 13–14).

The skills of the firm's workforce are the link to the projects that the firm can execute, because projects usually require specific skills. A project within the IT center can be the development of new features for a database that contains data on life insurance customers. The new features may allow sales personnel to record and access more information about their customers. This project may require three skills: experience in developing databases, knowledge of life insurance business, and finally the capability to hold training courses in which the sales personnel is instructed how to use the new features. Consequently, the project team must cover these three skills.

We distinguish four sets of projects. The first set is the set $\mathcal{P}^{\text{ongoing}}$ of ongoing projects, which the firm has already started but not completed before period $t = 1$. We assume that all started projects must be completed, i.e., ongoing projects are continued. The second set $\tilde{\mathcal{P}} = \{1, \ldots, \tilde{P}\}$ includes projects that can be selected for realization within the planning horizon. The third set $\mathcal{P}^{\text{must}}$ represents those projects that must be conducted by the firm within the planning horizon, e.g., for reasons of legal compliance or obligations to fulfill contracts. Finally, after projects have been selected, the set $\mathcal{P} = \{1, \ldots, P\}$ comprises those projects that will be executed during the planning horizon. Hence, $\mathcal{P}^{\text{ongoing}} \subseteq \mathcal{P}$, $\mathcal{P}^{\text{must}} \subseteq \mathcal{P}$, and $\mathcal{P} \setminus (\mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}}) \subseteq \tilde{\mathcal{P}}$ holds.

We assume that for each project $p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}}$ a rough schedule is given. This schedule specifies the start period $t_p^{\text{start}} \in \mathcal{T}$ and the finish period $t_p^{\text{finish}} \in \mathcal{T}$, which mark the set of periods $\mathcal{T}_p = \{t_p^{\text{start}}, \ldots, t_p^{\text{finish}}\}$ during which project p is executed.[2] Furthermore,

[2]For ongoing projects, we consider only the remaining part that must be performed in the planning horizon. Thus, we set $t_p^{\text{start}} := 1$ for all $p \in \mathcal{P}^{\text{ongoing}}$.

the rough schedule specifies a set $\mathcal{S}_p \subseteq \mathcal{S}$ of skills that are required by project p, and a skill requirement r_{pst} for each skill $s \in \mathcal{S}_p$ in each period $t \in \mathcal{T}_p$ of project execution. The skill requirements are also expressed in man-hours. A requirement $r_{pst} = 50$ is accomplished by a worker k in 50 hours if his skill level l_{ks} is equal to 1. For another worker k' with $l_{k's} = 2$ the same requirement would result in a workload of only 25 hours. The terms *skill requirement* and *project requirement* will be used interchangeably in the remainder of this thesis.

Let us briefly discuss whether the assumption is realistic that a rough schedule is given for projects. Proposals for projects may originate from inside the firm or from outside (cf. Zimmermann et al., 2010, pp. 2–3). Proposals from inside lead to *internal projects*, which often aim at improving processes within the firm. These proposals are usually put forward by the management or by a worker who is involved in the process. A source of proposals from outside the firm are potential customers who request a customized product. If such a proposal leads to an order, the fulfillment of this order constitutes an *external project*. For both internal and external projects the proposal must include a preliminary schedule or an initial work breakdown structure that enables managers to make an informed decision about whether to select the project or not (cf. Schwarze, 2001, pp. 54–55). A schedule as well as a work breakdown structure define work packages that must be delivered to complete a project (cf. Kerzner, 2013, pp. 529–536). They also state the estimated size of the work packages in man-hours and the order in which the work packages must be accomplished due to technical or logical restrictions. In our setting, the work packages are represented by the requirements r_{pst}. Furthermore, for external projects the start and completion date are often prescribed by the customer. In our setting, these prescribed dates correspond to the parameters t_p^{start} and t_p^{finish} of a project. For internal projects, it is more likely that the project start is not fixed, but can be postponed or brought forward to periods with low workload. A variable project start period could be easily integrated into our approach, e.g., Heimerl and Kolisch (2010a) present a model where projects can start within a time window.

Figure 3.1 illustrates the main characteristics of the setting of our problems. To ease reading, the notation in the figure is slightly changed: d_1 stands for department $d = 1$ and d_2 for department $d = 2$. The same holds for the other indices.

The skills link workers and projects. A skill s that is mastered by worker k and required by project p is termed "matching skill" between worker k and project p. Let $\mathcal{S}_{kp}^{\text{match}} = \mathcal{S}_k \cap \mathcal{S}_p$ denote the set of matching skills between worker k and project p. If $\mathcal{S}_{kp}^{\text{match}} \neq \emptyset$, worker k is called suitable for project p and project p is called suitable for worker k. Given the set $\mathcal{S}_{kp}^{\text{match}}$ for each pair $(k, p) \in \mathcal{K} \times \left(\mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}}\right)$, we can derive for each worker $k \in \mathcal{K}$ his set $\hat{\mathcal{P}}_k^{\text{suit}} = \left\{p \in \left(\mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}}\right) \;\middle|\; \mathcal{S}_{kp}^{\text{match}} \neq \emptyset\right\}$ of suitable projects and for each project p, $p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}}$, its set $\mathcal{K}_p^{\text{suit}} = \{k \in \mathcal{K} \mid \mathcal{S}_{kp}^{\text{match}} \neq \emptyset\}$ of suitable workers. Similarly, let the set $\mathcal{P}_k^{\text{suit}} = \{p \in \mathcal{P} \mid \mathcal{S}_{kp}^{\text{match}} \neq \emptyset\}$ contain the projects that were selected for the portfolio and that are suitable for worker k.

For each ongoing project $p \in \mathcal{P}^{\text{ongoing}}$, a project team exists already. This team comprises the workers that have already contributed to project p. Let the set $\mathcal{K}_p^{\text{assigned}}$ denote these workers that have already contributed to project p, i.e., that have already been assigned to project p. Other workers can join this team and existing team members need not contribute to project p during the planning horizon.

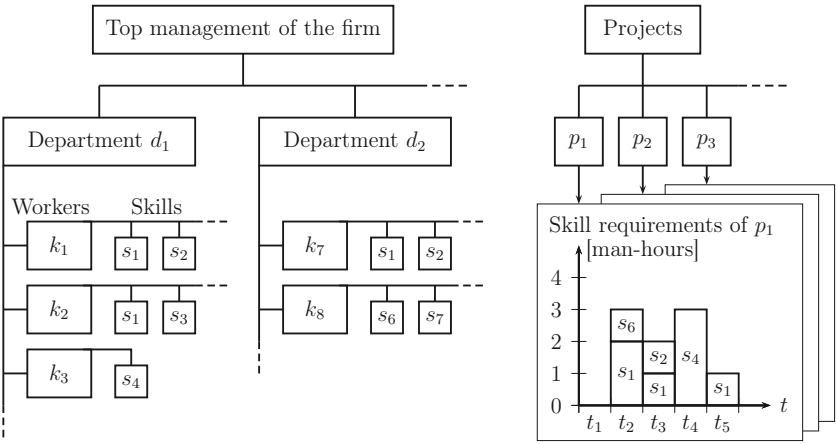


Figure 3.1: Main characteristics of the firm and the projects

Based on these characteristics of the firm and the projects, we outline our three problems in hierarchical order in the following three sections.

3.2 The project selection problem

The problem of project selection arises when the requirements of projects that the management deems beneficial for the firm exceed the availability of the firm's workforce. Then, the task of the management is to determine a project portfolio that provides maximum benefit among those project portfolios that are feasible with regard to workers' availabilities. We first look at portfolio membership and at the benefit of a portfolio and discuss assumptions that we make for the valuation of a portfolio. Eventually, we consider availabilities of workers in more detail.

To model the decision about membership of a project p in the project portfolio, we introduce the binary decision variable $z_p \in \{0,1\}$, $p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}}$, which equals 1 if project p is selected, and 0 otherwise. As already mentioned, there are not only projects that can be selected or not, but also ongoing projects, which were started in the past and which must be continued, and there are also mandatory projects, which must be conducted to fulfill contracts or to comply with new legal requirements. This means that the portfolio must contain all ongoing projects $p \in \mathcal{P}^{\text{ongoing}}$ and all mandatory projects $p \in \mathcal{P}^{\text{must}}$, thus $z_p = 1$ holds for $p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}}$. The actual selection concerns only the projects $p \in \tilde{\mathcal{P}}$, which can be picked for the portfolio. Though, to guarantee feasibility with respect to workers' availabilities, ongoing and mandatory projects must be taken into account when the portfolio is determined.

We assume that the benefit of each project $p \in \tilde{\mathcal{P}}$, e.g., the expected profit, is independent of the selection or rejection of other projects, that the benefit can be expressed

by a single parameter $b_p \in \mathbb{N} \setminus \{0\}$, and that a point estimate for b_p is given.[3] The more beneficial project p is, the higher is the value of the benefit parameter b_p. The benefit of a project portfolio can be calculated as the sum of the benefits of those projects $p \in \tilde{\mathcal{P}}$ that are part of the portfolio.

Our approach makes three strong assumptions, which simplify measuring the benefit of a portfolio:

(1) We assume that a project can be rated by the value of a single parameter. Though, usually there are multiple criteria such as strategic fit, urgency, or net present value that are considered when a project is assessed. Then, instead of a single parameter, there results a vector of parameters, which represents the benefit of a project.

(2) We assume that there is a single decision maker who rates each project and who decides about the portfolio. In many firms, however, there is not a single decision maker, but a group of managers who have to assess a project and whose assessments may differ from one another. Their different views result in different vectors of multiple parameters.

(3) Finally, we neglect interactions between projects. Interactions between two projects occur when both projects are selected. The interactions affect their parameters, e.g., their resource requirements or their benefits.

    Fox et al. (1984) distinguish three types of interactions that can occur between projects: resource, outcome, and benefit interactions. Resource interactions occur, for example, when two projects can share resources such that the costs for executing both projects are lower than the sum of the costs of the two projects when executed alone. Resource interactions also occur when resource requirements of a project depend on the selection of another project. Outcome interactions refer to the success probability of projects. The success probability of project p can depend on the success or failure of another project p'. Benefit interactions are subdivided into impact and present value interactions. Impact interactions exist, for example, when the product developed in a project competes with the product developed in another project and the sales of both products are less than the sum of individual sales which would have been realized for each product without competition. Present value interactions exist when the present value of the portfolio is not an additive function of the present values of the projects in the portfolio. Fox et al. show that present value interactions can occur even when no other interactions are present.

The assumptions impose limitations on the applicability of our approach. In the following, we report on approaches taken from literature that do without such limiting assumptions. Some of the approaches could be integrated into our approach in order to mitigate the described limitations.

ad (1) and (2): Lai and Xue (1999) present a procedure that transfers a multiple criteria, multiple decision maker problem within three steps into a single criterion, single

[3] For the case of uncertainty where a decision maker can specify for each parameter b_p, $p \in \tilde{\mathcal{P}}$, only an interval which contains the "true" value of b_p, Liesiö et al. (2007) outline an approach to select a robust portfolio. Here, a robust portfolio means a portfolio whose benefit is reasonably large for all realizations of the parameter values.

decision maker problem. In the first step, Lai and Xue basically average the assessments of all managers for each criterion. In the second step, the judgments of the managers on the importance of each criterion are also basically averaged. Finally, a linear programming approach is applied to determine a weight for each criterion such that the weights are "acceptable" for the managers. The procedure of Lai and Xue (1999) leads to a model that meets our first two strong assumptions, i.e., the assumption of a single parameter representing project benefit and the assumption of a single decision maker. Hence, their procedure could be applied to the projects $p \in \tilde{\mathcal{P}}$ in advance of our project selection.

ad (3): Schmidt (1993) presents a model that copes with all three types of interactions resulting in a nonlinear fourth order objective function with quadratic constraints. Santhanam and Kyparisis (1996) outline a nonlinear model that captures resource and benefit interactions. They linearize the model using the efficient technique of Glover and Woolsey (1974), which requires only continuous variables in addition to the binary selection variables and merely a small number of additional constraints. Eilat et al. (2006) propose a method for project selection that is based on the data envelope analysis. Their method takes into account project interactions, however, these interactions are considered not until candidate portfolios have been generated.

Our approach can be adapted to the method of Santhanam and Kyparisis (1996) without major effort. The methods of Schmidt (1993) and Eilat et al. (2006), in contrast, cannot be integrated into our approach without substantial changes of our approach. Hence, especially outcome interactions remain a severe limitation for our method and we must answer the question whether we can adhere to our method and when we should abandon it.

We adhere to our simple portfolio valuation for two reasons: First, we could adopt alternative approaches for valuation if necessary. The approach of Lai and Xue (1999) would just precede our approach, for example. Second, since our focus is on project teams, we are more interested in integrating the workers' availabilities during project selection such that the portfolio can be accomplished by the workforce. Benefit and outcome interactions are not critical for our focus on project teams. Anyway, outcome interactions are prevalent in a research and development environment where the success of a project is uncertain. In other environments, e.g., in an IT center, the probability of project failure is rather small, as project tasks are not utterly novel in most cases. Resource interactions, however, are important for our focus, because they can impact the skill requirements of projects. If such resource interactions exist, alternative approaches, e.g., the one of Santhanam and Kyparisis (1996), should be considered.

We do not only neglect interactions, but also interdependencies between projects. To explain the difference between interactions and interdependencies, we consider the case of two projects, which we denote by p and p'. Interactions affect the parameters of p and p', e.g., the resource requirements or the benefits, when both projects are selected. Interdependencies directly affect the selection variables z_p and z'_p, e.g., by mutual exclusiveness. Mutual exclusiveness means that either project p or project p' can be selected for the portfolio, but not both. Weingartner (1966) as well as Ghasemzadeh et al. (1999) take into account the same two types of interdependencies between projects by additional constraints. The first type of interdependencies is mutual exclusiveness of projects, the

second is a dependency that allows selection of project p' only if another project p is selected. The constraints for these two types of interdependencies could easily be adopted for our approach.

Having discussed project interdependencies and the three assumptions that we make, we will now describe three constraints for our project portfolio and begin with the skill requirements. For each project that is selected, we have to ensure that its requirements are satisfied. To this end, we introduce decision variables $\hat{y}_{kpst} \in \mathbb{R}_{\geq 0}$, $k \in \mathcal{K}$, $p \in \hat{\mathcal{P}}_k^{\text{suit}}$, $s \in \mathcal{S}_{kp}^{\text{match}}$, $t \in T_p$, which indicate the preliminary contribution of worker k to skill s of project p in period t. This contribution is expressed in man-hours. If $\hat{y}_{kpst} = 10$ and $l_{ks} = 2$, worker k contributes for 10 hours to project p and accomplishes 20 hours of the requirement r_{pst} of project p for skill s. Note that $\hat{y}_{kpst}$ is only a preliminary value for the contribution, because the actual contribution will be determined later when preferably small project teams are constructed.

We assume that a worker can contribute to different projects in each period $t \in \mathcal{T}$. A worker can perform different skills for one project in each period t. Furthermore, he can perform the same skill for more than one project in each period t and he can also contribute to different skills of different projects in every period. A requirement r_{pst} of project p for skill s in period t can be covered by an arbitrarily large number of workers or, more precisely, by up to $|\mathcal{K}_s|$ workers.

Apart from satisfying project requirements, we must observe two more restrictions, which both refer to the availabilities of workers. The first restriction refers to the availability of workers on the individual level. This restriction demands that the total contribution which a worker k makes across all projects and skills in period t must not exceed his availability R_{kt}. The second restriction refers to the availability of workers on the departmental level. In each period $t \in \mathcal{T}$, the total contribution to projects by workers who belong to the same department d must leave enough time for the workers to accomplish the departmental requirement rd_{dt}.

A solution for our problem of project selection entails values for the variables z_p, $p \in \tilde{\mathcal{P}}$, which reflect the completion of the portfolio, and values for the variables $\hat{y}_{kpst}$, $k \in \mathcal{K}$, $p \in \hat{\mathcal{P}}_k^{\text{suit}}$, $s \in \mathcal{S}_{kp}^{\text{match}}$, $t \in T_p$, which specify the allocation of project workload to workers. The values for the variables $\hat{y}_{kpst}$ imply project teams for each selected project $p \in \mathcal{P}$: All workers k with $\hat{y}_{kpst} > 0$ for at least one period $t \in \mathcal{T}_p$ and one skill $s \in \mathcal{S}_{kp}^{\text{match}}$ contribute to project p and are thus member of the team for project p. However, the project teams that result from solving the problem of project selection may be larger than necessary because the objective function of the selection problem does not penalize scattering the workload of a project across workers. After project selection, we try to minimize the average team size. This minimization problem is addressed in the next section, where we explicitly assign workers to projects and obtain final values for the contributions of workers to projects.

3.3 The workforce assignment problem

After having selected the project portfolio $\mathcal{P}$, we seek for a team of workers for each project $p \in \mathcal{P}$ and for an allocation of workload to the respective team members. Each project team must be able to cover the requirements of its project. Our aim is to minimize

the average team size over all projects $p \in \mathcal{P}$. In this section, we will state important assumptions underlying our problem, differentiate our aim from related but distinct objectives, and point to questionable implications of our objective. Finally, we introduce decision variables to formalize our problem and present the restrictions that constrain team formation and workload allocation in our setting.

The following assumptions are the same as for the project selection problem. We assume that a worker can contribute to different projects in each period $t \in \mathcal{T}$. A worker can perform different skills for one project in each period. Furthermore, he can perform the same skill for more than one project in each period and also different skills for different projects within any single period. A requirement r_{pst} of project p for skill s in period t can be covered by an arbitrarily large number of workers or, more precisely, by up to $|\mathcal{K}_s|$ workers.

Our objective for the assignment of workers to projects is to minimize average team size. This aim is equivalent to minimizing the total number of assignments of workers to projects, because the total number of assignments is equal to the average team size multiplied by the number of projects P, where P is a constant. Note that we limit neither the size of a team nor the number of assignments for a worker. It would be a different objective to minimize the maximum team size over all projects $p \in \mathcal{P}$. Another different objective would be to minimize the maximum number of projects which a worker is assigned to over all workers $k \in \mathcal{K}$. In general, different solutions are optimal for these related but distinct objectives.

The formulation of our objective implies that every assignment of a worker to a project has equal weight. Hence, it does not make a difference whether an additional worker is assigned to a project team of 3 workers or to a team of 78 workers. From the perspective of a single project, one may argue that it should make a difference because detrimental effects of an additional team member tend to decrease with team size. However, from the perspective of a single worker, there is no difference between being assigned to a large team or a small team, at least with respect to the extent of scattering the working time of this worker across projects. Different weights can favor solutions that feature more assignments than necessary. Since we want to avoid scattering workers across projects, we stick to the formulation of our objective, which features equal weights for all assignments.

To formalize our problem, we require two types of decision variables. The first type of variables is needed to count the assignments in the objective function of our problem. The second type of variables appears in the constraints only, these variables are needed to record the allocation of workload. In the objective function, we do not measure the average team size, but the total number of assignments of workers to projects. To record the number of assignments, we introduce binary decision variables $x_{kp} \in \{0, 1\}$, $k \in \mathcal{K}$, $p \in \mathcal{P}_k^{\text{suit}}$, which equal 1 if worker k is assigned to project p, and 0 otherwise. A worker k is assigned to project p if he contributes to project p. To record the contribution of worker k to project p for skill s in period t, we introduce the variable $y_{kpst} \in \mathbb{R}_{\geq 0}$, $k \in \mathcal{K}$, $p \in \mathcal{P}_k^{\text{suit}}$, $s \in \mathcal{S}_{kp}^{\text{match}}$, $t \in T_p$. Again, the contribution is expressed in man-hours. If $y_{kpst} = 10$ and $l_{ks} = 2$, worker k contributes for 10 hours to project p and accomplishes 20 hours of the requirement r_{pst} of project p for skill s in period t.

For a feasible assignment and workload allocation, three constraints have to be observed. (1) For each project $p \in \mathcal{P}$, its requirements must be satisfied by contributions from workers. (2) The sum of contributions of each worker $k \in \mathcal{K}$ across all

projects $p \in \mathcal{P}_k^{\text{suit}}$ must not exceed the worker's availability R_{kt} in each period $t \in \mathcal{T}$. (3) Finally, in each period $t \in \mathcal{T}$ and for each department $d \in \mathcal{D}$, the total contribution to projects by workers who belong to department d must leave enough time for the workers to accomplish the departmental requirement rd_{dt}.

For each ongoing project $p \in \mathcal{P}^{\text{ongoing}}$, a project team has already been formed in the past. This team of workers is denoted by the set $\mathcal{K}_p^{\text{assigned}}$. For each team member $k \in \mathcal{K}_p^{\text{assigned}}$, the variable x_{kp} is fixed to 1, indicating the existing assignment. These assignments, which have been made in the past, persist for the future, but we do not demand that a worker $k \in \mathcal{K}_p^{\text{assigned}}$ must contribute to project p in any period $t \in \mathcal{T}_p$, even if he could. Other workers, not yet assigned to project p, can join the existing team and accomplish requirements of project p. For each project p that is not an ongoing project, $\mathcal{K}_p^{\text{assigned}} = \emptyset$ holds as there exists no team for project p so far.

A solution for our problem of assigning workers to projects entails values for the variables x_{kp}, $k \in \mathcal{K}$, $p \in \mathcal{P}_k^{\text{suit}}$, which reflect the project teams, and values for the variables y_{kpst}, $k \in \mathcal{K}$, $p \in \mathcal{P}_k^{\text{suit}}$, $s \in \mathcal{S}_{kp}^{\text{match}}$, $t \in T_p$, which specify the allocation of project workload to workers. While the variables $\hat{y}_{kpst}$, which are obtained as a solution for the problem of selecting projects, provide preliminary values for the allocation of project workload, the values of y_{kpst} state the final allocation of project workload. This final allocation may result in an unbalanced utilization of workers. While one worker may have to perform 100 hours of project workload in a period, another worker may have to perform only 50 hours. To achieve a more even utilization of workers at least within departments, we can distribute the departmental workload of each department $d \in \mathcal{D}$ among the members of department d such that the resulting working times are better leveled. This leveling problem is addressed in the next section.

3.4 The utilization leveling problem

Workers must not only satisfy project requirements, but also departmental requirements. The time that worker k can accomplish departmental workload in period t is limited by his availability that remains from his initial availability R_{kt} after he accomplished project workload. To calculate the *remaining availability after project work* for a single worker k, we subtract the time that he must spend for performing project work from his initial availability R_{kt}. The solution to the problem of forming project teams and allocating project workload to workers guaranteed that the workers of each department $d \in \mathcal{D}$ have enough remaining availability after project work to accomplish the departmental workload rd_{dt} in all periods $t \in \mathcal{T}$. However, this solution did not specify how the departmental workload rd_{dt} of a period t is distributed among the workers of department d. Whether different allocations of the departmental workload are possible depends on the total remaining availability after project work for the workers in period t. If the total remaining availability after project work for the members of department d is equal to the departmental workload rd_{dt}, only one feasible allocation of departmental workload exists. If the total remaining availability is greater than rd_{dt}, different allocations of the workload are feasible, and we can use this degree of freedom to determine an allocation of departmental workload that levels the utilization of workers within a department as well as possible. A leveled utilization of workers is claimed by workers themselves on grounds

of fairness and it is a common aim for firms (cf. Lee and Ueng, 1999; Huang et al., 2006; Valls et al., 2009; and Eiselt and Marianov, 2008, for example). In this section, we will describe how utilization can be defined in our setting, outline the leveling problem, and critically reflect our assumptions about allocation of departmental workload.

Our definition of the utilization of a worker requires the hours worked by the worker. For the ease of presentation, let us assume that the hours worked by worker $k \in \mathcal{K}$ comprise the time he spends for projects and the time he spends for accomplishing departmental workload. This means that all workload that is allocated to worker k in period $t \in \mathcal{T}$ originates from projects and from his department only, i.e., there are no individual tasks assigned to worker k that reduce his initial availability. Hence, the availability R_{kt} of worker k in period t is equal to his initial availability.

The utilization of workers can be defined in two different ways. The first way defines the utilization of worker k in period t as the total hours worked by worker k in period t. The second way defines utilization as the ratio of total hours worked by worker k in period t to his availability R_{kt}. While the first definition of utilization considers the absolute time spent by a worker for projects and his department, the second definition considers the relative time spent by a worker, because the time spent is set in relation to the time available.

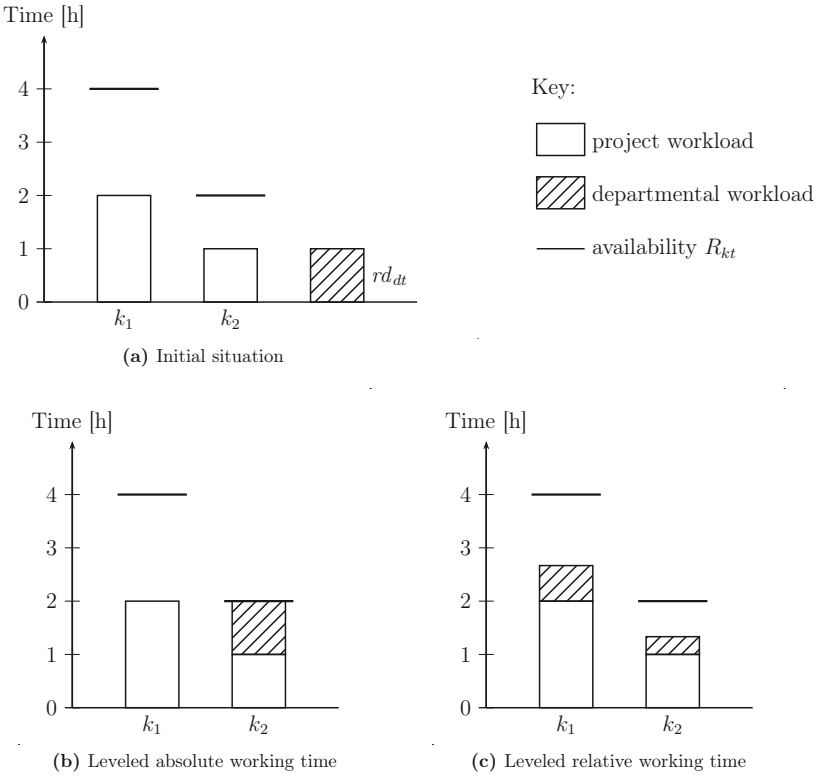
The problem of allocating departmental workload is illustrated in Figure 3.2. The initial situation in a period t is shown in Figure 3.2(a): Project workload has already been allocated to the workers k_1 and k_2, who are the only members of department d. They have an availability R_{kt} of four hours and two hours, respectively. The departmental requirement rd_{dt} that must be accomplished by k_1 and k_2 comprises one hour. Figure 3.2(b) depicts the allocation that levels the absolute working time of k_1 and k_2, while Figure 3.2(c) shows the allocation of the departmental workload where the relative working time is leveled. In the latter case worker k_1 has to accomplish 40 minutes and worker k_2 20 minutes of the departmental workload rd_{dt}.

As in our example in Figure 3.2, the two different definitions of utilization normally lead to different allocations of departmental workload if a leveled utilization is pursued. Though, from a computational point of view, both definitions are equivalent, as models and algorithms for the first definition can easily be adjusted to suit the second definition. Therefore, we will consider only the first definition, which allows a shorter notation.

To formulate the problem of allocating departmental workload, we introduce a decision variable $yd_{kt} \in \mathbb{R}_{\geq 0}$, $k \in \mathcal{K}$, $t \in \mathcal{T}$, that represents the time that worker k performs departmental workload in period t. We express the time yd_{kt} in hours. Remember that all department members perform departmental workload with the same efficiency, i.e., they all have an implicit "skill level" of 1 for performing workload of their department. If $yd_{kt} = 10$, worker k spends 10 hours for his department d and accomplishes 10 hours of the requirement rd_{dt} of department d in period t.

Our objective for the problem of allocating departmental workload is to level the total hours worked over all workers of each department $d \in \mathcal{D}$ in each period $t \in \mathcal{T}$. The total hours worked by worker k in period t are the sum of his contributions to projects and to his department. Leveling the total hours worked means that we want to find an allocation of departmental workload such that the total hours worked by all workers of a department are as equal as possible.

Two constraints must be observed. First, the total hours worked by a worker must not



(a) Initial situation

(b) Leveled absolute working time

(c) Leveled relative working time

Figure 3.2: The problem of allocating departmental workload in period t

exceed his availability. Second, the total workload allocated to workers of department d in period t must be equal to the requirement rd_{dt}.

Let us briefly elaborate on the assumptions for our leveling problem. Recall that we presume that the allocation of project workload is fixed and cannot be altered at this stage. We assume that the workload of a department is arbitrarily divisible and can thus be arbitrarily allocated to the workers of the department, as long as their availabilities are observed. Hence, it can happen that a departmental task that would be accomplished best by a single worker is split among three workers. This split causes a great need for coordination between those three workers. To avoid such an inefficient allocation of departmental workload, the head of the department should consider the solution of our leveling problem as a recommendation or as a point of reference that may require adjustment.

Having defined the problem of project selection, the problem of assigning workers

to projects and allocating project workload, and the problem of allocating departmental workload, we will formalize the three problem definitions by setting up optimization models in the next chapter.

Chapter 4

Optimization models and complexity analysis

In this chapter, we will present optimization models for the three problems that were outlined in the previous chapter. Furthermore, we will judge the complexity of the problems. In Section 4.1, we discuss the suitability of an integrated approach and of an alternative hierarchical planning approach for the three problems, i.e., for the project selection problem, the workforce assignment problem, and the utilization leveling problem. The hierarchical approach comprises three stages—one for each problem. We conclude that the hierarchical approach is preferable and present the corresponding optimization models for the three problems in Sections 4.2, 4.3, and 4.4, respectively. For our key problem of assigning workers to projects and allocating project workload, we will discuss limitations of our modeling approach in detail and point out potential remedies in Section 4.3. Additionally, we will present an integrated, monolithic optimization model in Section 4.5. The optimization models are mathematically precise statements of the problems. Each model features one objective function, various sets of constraints, and different sets of decision variables. The constraints define the solution space and allow to check if a solution, which is defined by the values of the decision variables, is feasible. The objective function allows to compare two solutions and to decide which of these two solutions is better. Finally, we will elaborate on the complexity of our three problems of the hierarchical planning approach in Section 4.6. The optimization models together with the evaluation of their complexity serve as a basis for the solution methods that will be presented in the next chapter.

4.1 An integrated approach vs. a hierarchical planning approach

In this section, we will briefly discuss two alternative approaches to tackle the three problems that were outlined in Sections 3.2–3.4. The first approach tries to simultaneously solve the three problems by formulating an integrated, monolithic optimization model. The second approach formulates separate optimization models, orders these models hierarchically, and solves one at a time. The separate models are only partially integrated. We will argue that the second approach is preferable for our problems and outline a three-stage hierarchical planning approach, which is partially integrated.

The first approach for our three problems, a monolithic model, integrates all decision variables into a single model. To integrate the three objectives of maximizing portfolio

M. Walter, *Multi-Project Management with a Multi-Skilled Workforce,* Produktion und Logistik, DOI 10.1007/978-3-658-08036-5_4, © Springer Fachmedien Wiesbaden 2015

benefit, minimizing average team size, and leveling working times into this single model, alternative roads can be selected (cf. Ehrgott, 2005; Neumann and Morlock, 2002, pp. 135–142; Domschke and Drexl, 2007, pp. 55-59). Two roads that are often selected are as follows. The first alternative associates a weight with each objective and considers the weighted sum of the objectives in a one-dimensional objective function. The second alternative considers a vector whose components are one-dimensional objective functions. For such a vector, a set of pareto-optimal solutions can be determined. The decision maker can select a solution out of this set of pareto-optimal solutions. For his selection, he must trade off the objectives against each other. For example, the decision maker could select the solution that offers the best leveled working times with an average team size of at most six workers per team and a portfolio benefit of at least 100.

In general, an integrated approach has two main disadvantages, which also become important in our case. First, an integrated approach tries to generate a detailed master plan and must, hence, process an enormous amount of data and information. In our case, a monolithic model would become very complex due to the high number of decision variables (cf. Günther, 1989, pp. 9–10). Second, a solution for the monolithic model fixes even those decision variables that lie in the distant future, although the situation in the distant future tends to be uncertain.

To overcome these two disadvantages of the first approach, the second approach structures the planning process hierarchically. The planning process is divided into several stages. The monolithic model is split into smaller problems, each problem is allocated to one stage of the planning process. The solution to a problem of a higher stage defines or constrains the solution space for problems at subordinated stages (cf. Schneeweiß, 1992, p. 13). An illustrative example of a hierarchical planning approach for a staffing problem is given by Grunow et al. (2004).

The hierarchy of planning problems often follows the importance of the corresponding decisions or the time horizon for which these decisions are made. The time horizon of a decision and its importance tend to be closely related: A long-term decision is usually a very important decision, i.e., a strategic decision, whereas short-term decisions, which affect only the near future, tend to have less impact on a firm. Accordingly, long-term or strategic planning, mid-term or tactical planning, and short-term or operational planning are distinguished.

Compared to an integrated approach, a hierarchical planning approach has advantages and drawbacks. On the one hand, a hierarchical planning approach is computationally better tractable and enables to postpone decisions of subordinate stages to times when uncertainty about data is resolved. On the other hand, a hierarchical approach requires the decision maker to rank the problems according to their importance before solutions to the problems will be generated. Here, the decision maker cannot trade off conflicting objectives as well as with the integrated approach. For instance, if the portfolio benefit is maximized first and the average team size is minimized afterwards, it may not be possible at the second stage to find a solution that offers an average team size of at most six workers per team.

For our three problems, however, a ranking of objectives in order of their importance stands out clearly. The decision about the project portfolio is the most important one for the firm, because selecting those projects which generate the highest total benefit is what matters most. For a given portfolio of projects the firm may wish an efficient

execution of projects. An efficient execution is facilitated by small project teams and by not scattering workers across a great number of projects. Therefore, minimizing the average size of project teams is the second most important objective. Finally, the aim of leveling hours worked by department members can be pursued when the other two goals have been achieved.

A ranking of objectives according to the time horizon of the corresponding decisions leads to the same order as a ranking in order of importance. The decision about the project portfolio affects the complete planning horizon $\{1, \ldots, T\}$ unless the set $\mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}}$ of projects can be partitioned into two non-empty subsets $\hat{\mathcal{P}}^1$ and $\hat{\mathcal{P}}^2$ such that there exists a period $t \in \mathcal{T} \setminus \{T\}$ with $t \geq t_p^{\text{finish}}$ for all $p \in \hat{\mathcal{P}}^1$ and $t < t_p^{\text{start}}$ for all $p \in \hat{\mathcal{P}}^2$ that divides the planning horizon into two separate planning horizons $\{1, \ldots, t\}$ and $\{t+1, \ldots, T\}$. The decision of assigning workers to projects affects also the complete planning horizon if the set $\mathcal{P}$ of projects cannot be partitioned into two sets $\mathcal{P}^1$ and $\mathcal{P}^2$, as just explained. Though, while the decision about the portfolio is almost irreversible, it is relatively easy and inexpensive to change the composition of the team for a project at later points in time, even after the start of the project. Hence, we can conclude that the time span that is affected by decisions about project teams is actually shorter than the time span that is affected by the decision about the project portfolio.

The time span that is affected by the problem of leveling hours worked by allocating departmental workload is shorter than the time spans affected by the decisions made for selection and team formation. The leveling problem must be solved for each department in each period $t \in \mathcal{T}$, hence, it is a short-term decision problem. At the time of project selection, the utilization of workers in a later period t is uncertain. Thus, it is reasonable to solve the leveling problem when utilization can be estimated more exactly, maybe two weeks before period t starts. It is not reasonable to solve this problem at the time when projects are selected, e.g., 10 months before period t starts.

In regard to the advantages of the hierarchical approach in general, and in regard to its suitability for our problems, we propose the following three-stage planning process. At the first stage, we solve the problem of selecting a project portfolio. At the second stage, we seek for the minimum number of assignments of workers to those projects that were selected at the first stage. Finally, we level the hours worked by department members in each period $t \in \mathcal{T}$ immediately before period t begins.

Although we have separated the solution process for our three problems into three stages, the problems are not fully separated but partially integrated. Here, partial integration means that we take into account subordinate problems when a problem of a superior stage is solved. When we solve the problem of project selection at the first stage, we take into account the availabilities of the workers who must accomplish the projects. We ensure that the requirements of the selected projects comply with the workers' availabilities. Furthermore, we take into account that sufficient availability remains for accomplishing departmental workloads. Also at the second stage, we take departmental workloads into account. For instance, at the time of project selection, the management of the firm may know that department d is busy with preparing a report every first month of a quarter and that department d' must prepare an exhibition appearance in period $t = 10$. These pieces of information can be integrated into the decision about the project portfolio.

In the subsequent sections we will present optimization models for the problem on each stage of our three-stage planning approach.

4.2 A model for the project selection problem

For a neat presentation of the model for project selection, we will introduce one further identifier. Recall that the set $\hat{\mathcal{P}}_k^{\text{suit}}$ is a subset of the union of the sets $\mathcal{P}^{\text{ongoing}}$, $\mathcal{P}^{\text{must}}$, and $\tilde{\mathcal{P}}$ and contains those projects p that are suitable for worker k. Let $\hat{\mathcal{P}}_k^{\text{suit}}(t) \subseteq \hat{\mathcal{P}}_k^{\text{suit}}$, $t \in \mathcal{T}$, denote the set of projects that are suitable for worker k and that are executed in period t. In other words, $\hat{\mathcal{P}}_k^{\text{suit}}(t)$ contains those projects p of the set $\hat{\mathcal{P}}_k^{\text{suit}}$ for which $t_p^{\text{start}} \leq t \leq t_p^{\text{finish}}$ holds.

Now, our problem of selecting projects can be modeled by (4.1)–(4.7). Model (4.1)–(4.7) is a mixed-integer linear programming (MIP) model with binary decision variables z_p, $p \in \tilde{\mathcal{P}}$, that indicate whether project p is selected or not, and with non-negative continuous decision variables $\hat{y}_{kpst}$, $k \in \mathcal{K}$, $p \in \hat{\mathcal{P}}_k^{\text{suit}}$, $s \in \mathcal{S}_{kp}^{\text{match}}$, $t \in \mathcal{T}_p$, that represent the workload that worker k performs for project p and skill s in period t.

$$\text{Max.} \quad \sum_{p \in \tilde{\mathcal{P}}} b_p z_p \tag{4.1}$$

$$\text{s. t.} \quad \sum_{k \in \mathcal{K}_s} (l_{ks} \hat{y}_{kpst}) = r_{pst} z_p \qquad p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}},\ s \in \mathcal{S}_p,\ t \in \mathcal{T}_p \tag{4.2}$$

$$\sum_{p \in \hat{\mathcal{P}}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} \hat{y}_{kpst} \leq R_{kt} \qquad k \in \mathcal{K},\ t \in \mathcal{T} \tag{4.3}$$

$$\sum_{k \in \mathcal{K}_d} \left(R_{kt} - \sum_{p \in \hat{\mathcal{P}}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} \hat{y}_{kpst} \right) \geq r d_{dt} \qquad d \in \mathcal{D},\ t \in \mathcal{T} \tag{4.4}$$

$$z_p = 1 \qquad p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \tag{4.5}$$

$$z_p \in \{0, 1\} \qquad p \in \tilde{\mathcal{P}} \tag{4.6}$$

$$\hat{y}_{kpst} \geq 0 \qquad k \in \mathcal{K},\ p \in \hat{\mathcal{P}}_k^{\text{suit}},\ s \in \mathcal{S}_{kp}^{\text{match}},\ t \in \mathcal{T}_p \tag{4.7}$$

Objective function (4.1) maximizes the benefit of the project portfolio. Constraint set (4.2) ensures that each requirement r_{pst} of project p is satisfied if project p is selected. The requirement r_{pst} for skill s in period t is satisfied by contributions of workers who master skill s. The coefficient l_{ks} takes into account that the workers $k \in \mathcal{K}_s$ master skill s at different levels.

Constraint set (4.3) guarantees that a worker k does not spend more time for projects in a period t than the time he is available in this period. Constraint set (4.4) assures that the workers of every department have enough remaining availability to accomplish the departmental workload in every period. On the left-hand side of Constraint (4.4), we calculate for each worker $k \in \mathcal{K}_d$ of department d the time that remains of his initial availability R_{kt} in period t when contributions to projects are considered. The total remaining time of all workers of department d must be large enough to cover the departmental workload.

Constraint set (4.5) fixes the variables z_p, $p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}}$, to 1, because we have to

include these projects in the portfolio. Constraint sets (4.6) and (4.7) state the domains of the actual decision variables.

By variables $\hat{y}_{kpst}$ in conjunction with Constraints (4.2)–(4.4), we model the allocation of workload and the availabilities for each worker explicitly. This is necessary because we consider cases where at least two workers master more than one skill each and where for at least one of these skills different levels are distinguished. Otherwise, i.e., especially if only homogeneous skill levels were considered, it would be possible to aggregate the capacity of the workers and to spare all variables $\hat{y}_{kpst}$. Though, such an aggregation would lead to a model of exponential size in the number of skills S (cf. Grunow et al., 2004, Section 3.3).

Grunow et al. (2004, Section 3.3) and Grunow et al. (2002, Section 4.2) show how capacities of multi-skilled resources can be aggregated if skill levels are homogenous. In the instances of their problems, the number of skills must have been relatively small so that model size did not became critical. We will give an example that shows how their aggregation could be applied in our case if skill levels were not differentiated. In this example, we also show why this aggregation is not applicable in the case of heterogeneous skill levels.

Example 4.1 Assume an instance A with $K = S = 2$, $\mathcal{T} = \{t\}$ and several projects that can be selected. Let $\mathcal{S}_{k_1} = \{s_1\}$ and $\mathcal{S}_{k_2} = \{s_1, s_2\}$ with $l_{ks} = 1$, $k \in \mathcal{K}$, $s \in \mathcal{S}_k$, and let $R_{kt} = 10$, $k \in \mathcal{K}$. Furthermore, assume that there is no departmental workload at all.[1] Let us denote the demand of the projects for skill $s \in \mathcal{S}$ that must be satisfied in period t by r_{st} where $r_{st} := \sum_{p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}} \mid t \in \mathcal{T}_p} r_{pst} z_p$. For instance A, Constraint sets (4.2) and (4.3) can then be replaced by the following constraints, which ensure that the skill requirements of all selected projects are satisfied and that the availabilities of all workers are regarded: $r_{s_1 t} \leq 20$, $r_{s_2 t} \leq 10$, and $r_{s_1 t} + r_{s_2 t} \leq 20$. In general, we would require that the following Constraint set must hold, where $\mathcal{S}'$ represents any non-empty subset of $\mathcal{S}$:

$$\sum_{s \in \mathcal{S}'} r_{st} \leq \sum_{k \mid \mathcal{S}_k \cap \mathcal{S}' \neq \emptyset} R_{kt} \qquad \mathcal{S}' \subseteq \mathcal{S},\, \mathcal{S}' \neq \emptyset,\, t \in \mathcal{T} \tag{4.8}$$

Since the number of non-empty subsets of $\mathcal{S}$ is equal to $2^{|\mathcal{S}|} - 1$, Constraint set (4.8) comprises an exponential number of constraints what makes this aggregation unattractive when there is a large number of skills.

Now, let us turn to the case of heterogeneous skill levels. Consider an instance B which is identical to instance A except that now $\mathcal{S}_{k_1} = \mathcal{S}_{k_2} = \{s_1, s_2\}$ with $l_{k_1 s_1} = 0.5$ and $l_{k_1 s_2} = l_{k_2 s_1} = l_{k_2 s_2} = 1$ holds. Here, the capacity of worker k_1 depends on the skill that she performs. We will consider two possibilities to adjust Constraints (4.8) to this situation. As we will see, both possibilities do not work. One possibility is to replace the right-hand side of Constraint set (4.8) by $\sum_{k \mid \mathcal{S}_k \cap \mathcal{S}' \neq \emptyset} (R_{kt} \cdot \max_{s \in \mathcal{S}_k \cap \mathcal{S}'} l_{ks})$. Then, we obtain the constraints $r_{s_1 t} \leq 15$, $r_{s_2 t} \leq 20$, and $r_{s_1 t} + r_{s_2 t} \leq 20$. According to these constraints, a project portfolio with $r_{s_1 t} = 15$ and $r_{s_2 t} = 1$ is a feasible solution of the selection problem, but there exists no corresponding feasible solution for the variables $\hat{y}_{kpst}$, i.e., there exists no feasible disaggregation. A second possibility is to replace the right-hand side of Constraint set (4.8) by $\sum_{k \mid \mathcal{S}_k \cap \mathcal{S}' \neq \emptyset} (R_{kt} \cdot \min_{s \in \mathcal{S}_k \cap \mathcal{S}'} l_{ks})$. This replacement yields the constraints $r_{s_1 t} \leq 15$, $r_{s_2 t} \leq 20$, and $r_{s_1 t} + r_{s_2 t} \leq 15$. These constraints render the

[1] This assumption is no loss of generality because we could interpret the requirements of each department as a project, as explained in Subsection 4.3.2 on page 64.

solution $r_{s_1t} = r_{s_2t} = 10$ infeasible, although there exists a disaggregation leading to feasible values for the variables $\hat{y}_{kpst}$. □

As Example 4.1 indicates, capacities of workers cannot be aggregated when skill levels are heterogeneous because the capacity of a worker depends on the skill or, to be more precise, on the skill mix which he performs. However, this skill mix is not known in advance, i.e., not before the model is solved. Hence, we must explicitly model the allocation of workload and the availabilities for each worker.

4.3 Models for the workforce assignment problem and their limitations

In this section, we consider models for the problem of assigning workers to projects and allocating project workload to workers. In Subsection 4.3.1, we will present two MIP models for this problem. The two models are alternative fomulations of the workforce assignment problem. Since this problem is in the focus of this thesis, we point out limitations of the models in Subsection 4.3.2 and outline potential extensions that mitigate these limitations.

4.3.1 Two alternative models for the workforce assignment problem

In this subsection, we will present two MIP models for the problem of assigning workers to projects and allocating project workload to workers. The first model is termed *standard model*, it can be intuitively derived from the problem definition in Section 3.3. The second model is named *network model*, because it implies for each period $t \in \mathcal{T}$ a network flow model. In such a network flow model, working time is assumed to flow from workers to projects and departments in order to cover their requirements.

To obtain a sparse formulation of both MIP models, we introduce one further identifier, analogously to the previous section. Remember that the set $\mathcal{P}_k^{\text{suit}} \subseteq \mathcal{P}$ includes those projects p that are suitable for worker k due to matching skills. Let $\mathcal{P}_k^{\text{suit}}(t) \subseteq \mathcal{P}_k^{\text{suit}}$, $t \in \mathcal{T}$, denote the set of projects that are suitable for worker k and that are carried out in period t. Put another way, $\mathcal{P}_k^{\text{suit}}(t)$ contains those projects p of the set $\mathcal{P}_k^{\text{suit}}$ for which $t_p^{\text{start}} \leq t \leq t_p^{\text{finish}}$ holds.

The standard model is given by (4.9)–(4.16). The decision variables of the standard model are the binary variables x_{kp}, $k \in \mathcal{K}$, $p \in \mathcal{P}_k^{\text{suit}}$, which indicate whether worker k is assigned to project p or not, and the non-negative continuous variables y_{kpst}, $k \in \mathcal{K}$, $p \in \mathcal{P}_k^{\text{suit}}$, $s \in \mathcal{S}_{kp}^{\text{match}}$, $t \in \mathcal{T}_p$, which record the workload that worker k accomplishes for project p and skill s in period t.

$$\text{Min.} \quad \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}_k^{\text{suit}}} x_{kp} \tag{4.9}$$

$$\text{s. t.} \quad \sum_{k \in \mathcal{K}_s} (l_{ks} y_{kpst}) = r_{pst} \qquad p \in \mathcal{P},\, s \in \mathcal{S}_p,\, t \in \mathcal{T}_p \tag{4.10}$$

$$\sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst} \leq R_{kt} x_{kp} \qquad p \in \mathcal{P},\, k \in \mathcal{K}_p^{\text{suit}} \setminus \mathcal{K}_p^{\text{assigned}},\, t \in \mathcal{T}_p \tag{4.11}$$

$$\sum_{p \in \mathcal{P}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst} \leq R_{kt} \qquad k \in \mathcal{K},\, t \in \mathcal{T} \tag{4.12}$$

$$\sum_{k \in \mathcal{K}_d} \left(R_{kt} - \sum_{p \in \mathcal{P}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst} \right) \geq rd_{dt} \qquad d \in \mathcal{D},\, t \in \mathcal{T} \tag{4.13}$$

$$x_{kp} = 1 \qquad p \in \mathcal{P}^{\text{ongoing}},\, k \in \mathcal{K}_p^{\text{assigned}} \tag{4.14}$$

$$x_{kp} \in \{0,1\} \qquad p \in \mathcal{P},\, k \in \mathcal{K}_p^{\text{suit}} \setminus \mathcal{K}_p^{\text{assigned}} \tag{4.15}$$

$$y_{kpst} \geq 0 \qquad k \in \mathcal{K},\, p \in \mathcal{P}_k^{\text{suit}},\, s \in \mathcal{S}_{kp}^{\text{match}},\, t \in \mathcal{T}_p \tag{4.16}$$

Objective function (4.9) minimizes the total number of assignments of workers to projects and, thus, the average team size. The total number of assignments includes the assignments that have already been made for ongoing projects. Constraints (4.10) ensure that the requirement r_{pst}, $p \in \mathcal{P}$, $s \in \mathcal{S}_p$, $t \in \mathcal{T}_p$, is satisfied by contributions from workers who master skill s. The coefficient l_{ks} takes into account that these workers $k \in \mathcal{K}_s$ master skill s at different levels.

Constraints (4.11) link the variables x_{kp} and y_{kpst}. These constraints guarantee that worker k can only contribute to project p if he is assigned to project p. A contribution $y_{kpst} > 0$ for any skill $s \in \mathcal{S}_{kp}^{\text{match}}$ in any period $t \in \mathcal{T}_p$ requires $x_{kp} = 1$. Simultaneously, Constraints (4.11) force $x_{kp} = 1$ if worker k contributes to project p for any skill $s \in \mathcal{S}_{kp}^{\text{match}}$ in any period $t \in \mathcal{T}_p$. Hence, if worker k contributes to project p, he is automatically assigned to project p.

Constraints (4.11) are so called "big-M constraints", which allow to model logical conditions (cf. Bosch and Trick, 2005, pp. 77–78; Williams, 1999, pp. 154–160). In a general form, the right-hand side of Constraints (4.11) would be written as Mx_{kp}, where M is a sufficiently large constant. We chose R_{kt} for M, as R_{kt} is an upper bound for $\sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst}$ and, hence, sufficiently large.

Constraints (4.12) take care that the working time which worker k spends for projects in period t does not exceed his availability R_{kt}. Constraints (4.13) ensure for each department $d \in \mathcal{D}$ that the remaining availabilities of all workers of department d are large enough in every period $t \in \mathcal{T}$ to accomplish the departmental workload rd_{dt}.

For each ongoing project $p \in \mathcal{P}^{\text{ongoing}}$, a team $\mathcal{K}_p^{\text{assigned}}$ of workers exists already. For each member k of this team, Constraint set (4.14) fixes the corresponding variable x_{kp} to 1. Constraint sets (4.15) and (4.16) state the domains of the actual decision variables.

Let us briefly discuss the big-M constraints (4.11). We could have replaced Constraints (4.11) by

$$y_{kpst} \leq R_{kt} x_{kp} \qquad p \in \mathcal{P},\, k \in \mathcal{K}_p^{\text{suit}} \setminus \mathcal{K}_p^{\text{assigned}},\, s \in \mathcal{S}_{kp}^{\text{match}},\, t \in \mathcal{T}_p \tag{4.17}$$

or by

$$\sum_{s \in \mathcal{S}_{kp}^{\text{match}}} \sum_{t \in \mathcal{T}_p} y_{kpst} \leq \left(\sum_{t \in \mathcal{T}_p} R_{kt} \right) x_{kp} \qquad p \in \mathcal{P},\, k \in \mathcal{K}_p^{\text{suit}} \setminus \mathcal{K}_p^{\text{assigned}}. \tag{4.18}$$

To assess the three alternatives (4.11), (4.17), and (4.18), we scrutinize the number of constraints resulting from each constraint set and the tightness of each constraint set for the linear programming (LP) relaxation of the corresponding MIP. The LP relaxation of the standard model is obtained when the binary variables $x_{kp} \in \{0,1\}$ are replaced by continuous variables $x_{kp} \in [0,1]$. This relaxation is commonly used to solve the model by branch-and-bound or branch-and-cut methods. The higher the number of constraints which define the feasible region of the LP relaxation, the more time is generally required for solving the LP relaxation. The tighter the relaxation, the closer does the feasible region of the relaxation come to the convex hull of feasible integer points of the MIP. The tightness of the relaxation is vitally important and more important than the number of constraints. In general, the drawback of additional constraints is outweighed by far if these constraints tighten the relaxation (cf. Williams, 1999, pp. 190–197).

The largest number of constraints exhibits Constraint set (4.17). It comprises $\overline{|\mathcal{S}^{\text{match}}|}$ times as much constraints as Constraint set (4.11), where $\overline{|\mathcal{S}^{\text{match}}|}$ denotes the average number of matching skills between a project $p \in \mathcal{P}$ and a worker $k \in \mathcal{K}_p^{\text{suit}} \setminus \mathcal{K}_p^{\text{assigned}}$. The smallest number of constraints are contained in Constraint set (4.18), where a constraint for project p covers the whole duration of project p. Constraint set (4.11) has $\overline{|\mathcal{T}^{\text{proj}}|}$ times as much constraints as Constraint set (4.18), where $\overline{|\mathcal{T}^{\text{proj}}|}$ denotes the average duration of a project $p \in \mathcal{P}$.

We will use the following example to gain insight in the tightness of the three constraint sets:

Example 4.2 Consider worker k and project p with $\mathcal{S}_{kp}^{\text{match}} = \{s_1, s_2\}$, $\mathcal{T}_p = \{t_1, t_2\}$, $R_{kt_1} = 50$, and $R_{kt_2} = 100$. Let $y_{kps_1t_1} = y_{kps_2t_1} = 25$ and $y_{kps_1t_2} = y_{kps_2t_2} = 0$ be a feasible solution for the MIP. Then $x_{kp} \geq 1$ satisfies (4.11), $x_{kp} \geq 0.5$ satisfies (4.17) and $x_{kp} \geq \frac{1}{3}$ satisfies (4.18) in the LP relaxation of the corresponding MIP. □

In Example 4.2, Constraint set (4.11) is the tightest out of the three alternatives. Indeed, Constraint set (4.11) is always at least as tight as Constraint sets (4.17) and (4.18) and tighter than (4.17) and (4.18) in general.

From our considerations we concluded that Constraint set (4.11) is the best choice. Numerical tests supported our conclusion. These numerical tests are presented in Section 7.3.[2]

The network model is an alternative way to represent the same problem as the standard model. Since the properties of network structures can often be exploited to design efficient solution methods, it seems worthwhile to pursue this alternative approach. The network model regards in each period $t \in \mathcal{T}$ each worker $k \in \mathcal{K}$ as a source of working time. This source is represented by a node in the network model that supplies an amount of R_{kt} hours of working time. Projects and departments are regarded as sinks, i.e., as nodes that ask for working time. Project p asks for r_{pst} hours of working time for each skill $s \in \mathcal{S}_p$ in period t. Department d demands rd_{dt} hours of working time in period t. In the network of period t, working time can flow from source nodes along arcs via intermediate nodes to the sinks which represent the demand nodes. In any period $t \in \mathcal{T}_p$, working time can flow from worker $k \in \mathcal{K}_p^{\text{suit}}$ to project p only if worker k is assigned to project p. Our aim is to determine the minimum number of assignments of workers to projects that allow a flow of working time which satisfies all demands of projects and departments in every period.

[2]For a tighter formulation of the big-M Constraints (4.11) and (4.17) see Subsection 6.1.1.

The underlying network of a period is sketched in Figure 4.1, which also depicts additional flow variables, which are required for the network model. For Figure 4.1, we assume that projects p_1 and p_2 are executed in period t and that workers k_1 and k_2 belong to department d_1. Furthermore, we assume that worker k_1 and k_2 master the skills s_1 and s_2, which are required by project p_1. Note that Figure 4.1 illustrates only a section of the total network of period t in order to clarify the concept. A demand for working time is represented by a negative supply.

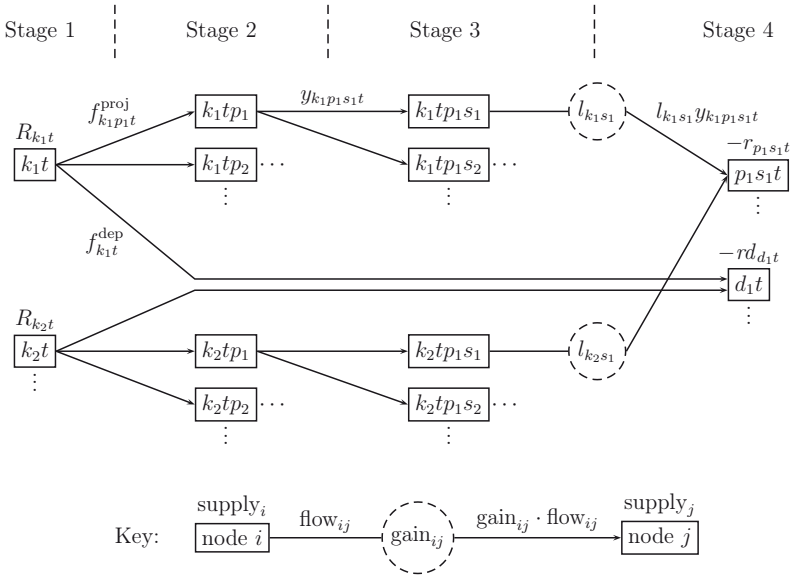


Figure 4.1: Section of the underlying network flow model for period t

Like the network of period t in Figure 4.1, the network of each period $t \in \mathcal{T}$ comprises four stages, with supply nodes at the first stage and demand nodes at the fourth stage. At the first stage of the network, the source nodes represent the workers $k \in \mathcal{K}$, who supply a flow of R_{kt} hours of working time. This flow from worker k is divided into flows f^{proj}_{kpt} to projects $p \in \mathcal{P}^{\text{suit}}_k(t)$ at the second stage and into a flow f^{dep}_{kt} to the department to which worker k belongs.

The flow f^{proj}_{kpt} to project p is split up into flows y_{kpst}, $s \in \mathcal{S}^{\text{match}}_{kp}$, to project demand nodes on the final stage. Before the flow y_{kpst} reaches the sink node that represents the project requirement r_{pst}, the flow is multiplied by a gain of l_{ks}. This gain weights the time that worker k spends for skill s of project p with his skill level l_{ks}. Since $l_{ks} \neq 1$ is possible, we obtain a network with gains (cf. Ahuja et al., 1993, pp. 566–568; Bertsekas, 1998, pp. 360–365). Problems on networks with gains are termed *generalized network problems*.

The network model requires two types of additional variables, which have already been introduced in Figure 4.1. First, variable $f^{\text{proj}}_{kpt} \in \mathbb{R}_{\geq 0}$, $k \in \mathcal{K}$, $p \in \mathcal{P}^{\text{suit}}_k$, $t \in \mathcal{T}_p$, records the flow between stage 1 and 2 from worker k to project p in period t. Second, the variable $f^{\text{dep}}_{kt} \in \mathbb{R}_{\geq 0}$, $k \in \mathcal{K}$, $t \in \mathcal{T}$, represents the flow between stage 1 and 4 from worker k to his department d in period t.

For each worker $k \in \mathcal{K}$ and all projects $p \in \mathcal{P}^{\text{suit}}_k$, the flow variables f^{proj}_{kpt} of all periods $t \in \mathcal{T}_p$ and hence the networks of all these periods are coupled by the binary decision variable x_{kp}. Each variable f^{proj}_{kpt}, $t \in \mathcal{T}_p$, must equal 0 if worker k is not assigned to project p, i.e., if $x_{kp} = 0$.

Although additional variables are required for the network model, it is worthwhile to consider this model, because underlying network structures can often be exploited by specialized network algorithms. These algorithms facilitate an efficient solution of the underlying problems (cf. Ahuja et al., 1993, pp. 402–403, for example).

The network model is given by (4.19)–(4.29). The decision variables of the network model are the binary variables x_{kp}, $k \in \mathcal{K}$, $p \in \mathcal{P}^{\text{suit}}_k$, which indicate whether worker k is assigned to project p or not, and the non-negative continuous variables y_{kpst}, $k \in \mathcal{K}$, $p \in \mathcal{P}^{\text{suit}}_k$, $s \in \mathcal{S}^{\text{match}}_{kp}$, $t \in \mathcal{T}_p$, which record the workload that worker k performs for project p and skill s in period t. Auxiliary variables are the non-negative continuous variables f^{proj}_{kpt}, $k \in \mathcal{K}$, $p \in \mathcal{P}^{\text{suit}}_k$, $t \in \mathcal{T}_p$, which represent the flow from worker k to project p in period t, and the non-negative continuous variables f^{dep}_{kt}, $k \in \mathcal{K}$, $t \in \mathcal{T}$, which represent the flow from worker k to his department in period t.

$$\text{Min.} \quad \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}^{\text{suit}}_k} x_{kp} \tag{4.19}$$

$$\text{s. t.} \quad R_{kt} = f^{\text{dep}}_{kt} + \sum_{p \in \mathcal{P}^{\text{suit}}_k(t)} f^{\text{proj}}_{kpt} \qquad k \in \mathcal{K},\, t \in \mathcal{T} \tag{4.20}$$

$$f^{\text{proj}}_{kpt} = \sum_{s \in \mathcal{S}^{\text{match}}_{kp}} y_{kpst} \qquad k \in \mathcal{K},\, p \in \mathcal{P}^{\text{suit}}_k,\, t \in \mathcal{T}_p \tag{4.21}$$

$$\sum_{k \in \mathcal{K}_s} (l_{ks} y_{kpst}) = r_{pst} \qquad p \in \mathcal{P},\, s \in \mathcal{S}_p,\, t \in \mathcal{T}_p \tag{4.22}$$

$$\sum_{k \in \mathcal{K}_d} f^{\text{dep}}_{kt} \geq r d_{dt} \qquad d \in \mathcal{D},\, t \in \mathcal{T} \tag{4.23}$$

$$f^{\text{proj}}_{kpt} \leq R_{kt} x_{kp} \qquad p \in \mathcal{P},\, k \in \mathcal{K}^{\text{suit}}_p \setminus \mathcal{K}^{\text{assigned}}_p,\, t \in \mathcal{T}_p \tag{4.24}$$

$$x_{kp} = 1 \qquad p \in \mathcal{P}^{\text{ongoing}},\, k \in \mathcal{K}^{\text{assigned}}_p \tag{4.25}$$

$$x_{kp} \in \{0, 1\} \qquad p \in \mathcal{P},\, k \in \mathcal{K}^{\text{suit}}_p \setminus \mathcal{K}^{\text{assigned}}_p \tag{4.26}$$

$$f^{\text{proj}}_{kpt} \geq 0 \qquad k \in \mathcal{K},\, p \in \mathcal{P}^{\text{suit}}_k,\, t \in \mathcal{T}_p \tag{4.27}$$

$$f^{\text{dep}}_{kt} \geq 0 \qquad k \in \mathcal{K},\, t \in \mathcal{T} \tag{4.28}$$

$$y_{kpst} \geq 0 \qquad k \in \mathcal{K},\, p \in \mathcal{P}^{\text{suit}}_k,\, s \in \mathcal{S}^{\text{match}}_{kp},\, t \in \mathcal{T}_p \tag{4.29}$$

Objective function (4.19) minimizes the total number of assignments and, hence, the average team size. The total number of assignments includes the assignments that have already been made for ongoing projects.

Constraints (4.20)–(4.22) are flow conservation constraints. Constraints (4.20) demand that the flow of working time from worker k to his department and to projects in period t must equal the supply R_{kt} of working time in period t. Constraints (4.21) ensure that the working time which worker k spends for all skills $s \in \mathcal{S}_{kp}^{\text{match}}$ of project p in period t equals the flow from worker k to project p in period t. Constraints (4.22) assure that requirement r_{pst} of project p for working time concerning skill s in period t is covered by contributions from suitable workers k. Their contributions are weighted with their skill levels l_{ks}.

Constraints (4.23) guarantee that the flow of working time from those workers who belong to department $d \in \mathcal{D}$ to their department d in period t is sufficiently large to cover the requirement rd_{dt}. Constraint set (4.23) allows that an excessive supply of working time is absorbed by the departments. As a consequence, the variable f_{kt}^{dep} does not specify the departmental workload that worker k must accomplish in period t, but is only an upper bound on the departmental workload that must be accomplished by worker k in period t.

Constraints (4.24) link the variables x_{kp} and f_{kpt}^{proj}. These constraints guarantee that worker k can only contribute to project p if he is assigned to project p. A contribution $f_{kpt}^{\text{proj}} > 0$ in any period $t \in \mathcal{T}_p$ and, hence, a contribution $y_{kpst} > 0$ for any skill $s \in \mathcal{S}_{kp}^{\text{match}}$ requires $x_{kp} = 1$. Simultaneously, Constraints (4.24) force $x_{kp} = 1$ if worker k contributes to project p in any period $t \in \mathcal{T}_p$. Thus, if worker k contributes to project p, he is automatically assigned to project p.

For each ongoing project $p \in \mathcal{P}^{\text{ongoing}}$, a team $\mathcal{K}_p^{\text{assigned}}$ of workers exists already. For each member k of this team, Constraint set (4.25) fixes the corresponding variable x_{kp} to 1. Constraint sets (4.26)–(4.29) state the domains of the actual decision variables. The decision variables whose domains are defined in (4.27) and (4.28) can be considered as auxiliary variables, which are required to model the network flows in each period $t \in \mathcal{T}$.

Constraints (4.24) are big-M constraints. They are equivalent to the big-M Constraints (4.11) of the standard model. Constraint set (4.24) can also be replaced by Constraint sets (4.17) or (4.18).

4.3.2 Limitations of the assignment models and potential remedies and extensions

We have already discussed limitations of our approach to project selection in Section 3.2 and limitations of our approach to leveling hours worked by allocating departmental workload in Section 3.4. Now, we will elaborate on limitations of the two models that we introduced in the previous Subsection 4.3.1. Both models have identical scope and seek for a solution to the same problem. They search for an assignment of workers to projects and for an allocation of project workload to workers. Since this problem is in the focus of this thesis, we dedicate a separate subsection to limitations of our models for this problem. The limitations that we discuss are the neglect of overtime, the neglect of learning effects[3], the neglect of the role of project managers, and the neglect of worker compatibility.

In many firms, workers are allowed to work overtime. If a worker works overtime, his regular availability is exceeded. Usually, the amount of extra hours per period is

[3]The consideration of learning effects would result in dynamic skill levels.

limited. Firms prescribe extra hours to meet peak demands. Workers are compensated for extra hours either by monetary means or by days off in other periods. If days off are granted instead of monetary rewards, many employment contracts prescribe that the average hours worked per period must not exceed the average regular availability per period. This means that extra hours must be completely compensated in the course of a year or so.

Our models (4.9)–(4.16) and (4.19)–(4.29) do not consider overtime. They regard the regular availability R_{kt} of worker k in period t as a hard constraint, which must not be violated. Hence, our modeling approach does not meet the conditions that prevail in many firms.

In the project management literature, overtime is rarely considered in models. A model that takes overtime into account is the model of Heimerl and Kolisch (2010a). Heimerl and Kolisch (2010a) want to minimize costs for wages. They associate wage rates with extra hours that are higher than wage rates for regular hours. In their model, extra hours are explicitly registered by distinct variables.

The integration of overtime into our models is possible as well. If extra hours are compensated by additional payments and need not be balanced over the planning horizon but are limited for each period, then we would just have to increase the availability R_{kt}. For example, assume that the regular availability of worker k in period t is given by $R_{kt} = 160$ and that 20 extra hours are allowed for worker k in period t, then we would set $R_{kt} := 180$. Since we do not consider variable costs, nothing else must be done. If we want to take variable costs for overtime into account, additional variables are necessary to record the extra hours of each worker in each period (cf. Heimerl and Kolisch, 2010a).

If extra hours must be compensated by days off in the course of the planning horizon, our models require three changes. First, we have to increase R_{kt}, $k \in \mathcal{K}$, $t \in \mathcal{T}$, by the number of extra hours that are allowed per period. For example, if the regular availability of worker k is given by $R_{kt} = 160$ for each period $t \in \mathcal{T}$ and if 20 extra hours were allowed per period, we would set $R_{kt} := 180$ for each period $t \in \mathcal{T}$.

Second, a new set of constraints must be added to our models. These constraints must demand that the number of hours worked by worker $k \in \mathcal{K}$ during the whole planning horizon does not exceed the number of regular hours that the labor contract allows for the planning horizon. In our example, worker k would not be allowed to work more than $160 \cdot T$ hours during the planning horizon $\mathcal{T} = \{1, \ldots, T\}$. Constraint (4.30) imposes this limit on the total working time of worker k during the planning horizon.

$$\sum_{t \in \mathcal{T}} \sum_{p \in \mathcal{P}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst} \leq 160 \cdot T \tag{4.30}$$

Third, the workload of each department must be interpreted as a project. The first two changes, which we presented for the standard model and the network model, are not in line with Constraints (4.13) and Constraints (4.23), respectively, which ensure that the remaining availability after project work is sufficiently large to cover departmental workloads in each period. The issue is that Constraint (4.30) does not register the time that worker k spends for departmental work. To fix this issue, the workload of each department $d \in \mathcal{D}$ has to be interpreted as a project. This interpretation results in D additional projects. The project that corresponds with department d has a requirement of rd_{dt} man-hours in period $t \in \mathcal{T}$. This workload can only be accomplished by members of

department d. Thus, department membership must be interpreted as a skill, resulting in D additional skills. Such an additional skill is mastered only by the members of the corresponding department. The skill level with which each member masters his departmental skill is equal to 1. Additional binary assignment variables that indicate whether worker k is assigned to the project that corresponds to his department or not, can be fixed to 1. These three changes of our models would allow to integrate overtime that is compensated by days off.

A consideration of overtime in a model raises the question how worked extra hours should be taken into account in the objective function of the model. If a firm grants monetary compensation for extra hours, it seems natural to minimize the payments for overtime. Though it is difficult to integrate this cost objective and the objective of a minimum average team size into a single objective function, because both objectives are expressed in different units. Natural weights for the two objectives are not apparent. Hence, it would make more sense to consider both objectives separately and to determine a set of pareto-optimal solutions. If the firm compensates extra hours by days off, the same difficulties arise if the firm wants to minimize the number of extra hours in order to achieve leveled working times for its workforce. If the firm does not aim at minimizing extra hours, extra hours need not be taken into account in an objective function.

We do not consider overtime in our models for two reasons. First, extra hours are rather an ad hoc measure to cover unforeseen workload peaks. Although flexible working time agreements have become more common, especially labor unions and works councils urge the management of firms to stick to regular working times and are opposed to the planned use of extra hours. Therefore, overtime is often not deemed a suitable way to expand capacity or to form small teams. The second reason for neglecting overtime is owed the novelty of our approach. Since this approach is the first of its kind, we restrict our models to the most essential elements and parts of the underlying problem in order to get good insight into basic properties of the problem. Effects that appear when input data or solution methods are changed stand out more clearly in case of a plain and compact model.

Furthermore, our models assume that the skill levels l_{ks}, $k \in \mathcal{K}$, $s \in \mathcal{S}_k$, are static. We ignore that skill levels might change due to effects of learning and forgetting. Learning and forgetting a skill is closely linked to performing the skill. When skill s is performed by worker k, the learning effect increases the skill level l_{ks}. The increase can be derived from a nonlinear learning curve (cf. Wright, 1936; Yelle, 1979). Forgetting appears when skill s is not performed for some time and decreases the skill level (cf. Chen and Edgington, 2005, p. 287).

In the literature, models exist that incorporate learning and forgetting (cf. Wu and Sun, 2006; Gutjahr et al., 2008, 2010; and Heimerl and Kolisch, 2010b, for example). These models apply nonlinear expressions to integrate the concept of the learning curve. The resulting models feature dynamic skill levels. Some of the models aim at allocating workload such that targets for skill levels are met at the end of the planning horizon.

The integration of dynamic skill levels into our models would require significant changes, which lead to nonlinear models. Though, in our opinion, the merit of considering dynamic skills is small in our case. We argue that the merit is small for two reasons. First, it is difficult and cumbersome to derive learning rates for each worker and each skill, especially because for some skills it can be costly to measure skill levels

at short intervals to construct a learning curve. Estimations of learning rates tend to be error-prone. Secondly, the typical length of our planning horizon of one year is relatively short compared to the duration of project tasks, which are quite complex and can last several months. Thus, learning effects within some periods should be rather small, as the number of units of output is small.[4] That is why we recommend to use static skill levels and evaluate skill levels once a year. Then, models for the next planning horizon can be fed with updated skill levels. Periodically updates of skill levels are also used by Süer and Tummaluri (2008).

If a firm aims at allocating project workload such that skills of workers are developed to meet skill level targets at the end of the planning horizon, a model featuring dynamic skill levels is advantageous. Nevertheless, if the firm wants that worker k gathers experience in skill s, we could simply add a constraint to our models. Assume that worker k is said to perform at least 100 hours of work for skill s during the planning horizon, then Constraint (4.31) would guarantee this experience.

$$\sum_{t \in \mathcal{T}} \sum_{p \in \mathcal{P}_k^{\text{suit}}(t) \,|\, s \in \mathcal{S}_p} y_{kpst} \geq 100 \tag{4.31}$$

The third limitation concerns the role of the project manager. Usually, each project has one project manager. She is the head of the project team, coordinates the team members, and is responsible for a successful implementation of the project (Kerzner, 2013, pp. 14–15).

While our models do not take the role of project managers into account, Yoshimura et al. (2006) and Patanakul et al. (2007) published models that explicitly consider the role of the project manager.

For our approach, we can imagine at least two ways to assign exactly one project manager to each project. First, project managers might be assigned in advance, i.e., before the assignment model is solved that determines the project teams. Second, the task of assigning one project manager to each project could be integrated into our assignment models. We would define a set $\mathcal{K}_p^{\text{PM}} \subseteq \mathcal{K}_p^{\text{suit}}$, $p \in \mathcal{P}$, of workers that are qualified to act as project manager for project p. Then, Constraint (4.32) would ensure that at least one worker out of the set $\mathcal{K}_p^{\text{PM}}$ is selected for managing project p.

$$\sum_{k \in \mathcal{K}_p^{\text{PM}}} x_{kp} \geq 1 \tag{4.32}$$

If in a solution more than one worker out of the set $\mathcal{K}_p^{\text{PM}}$ was assigned to project p, one of them must be selected as project manager. If it is not deemed suitable to include more than one potential project manager into a project team, we could demand that the left-hand side of Constraint (4.32) must equal 1.

Project managers have to accomplish special tasks for their project, e.g., administrative tasks. To model these requirements we could associate a distinct skill s with the

[4] It may even be difficult to define an appropriate unit of output. Though, a definition of a unit of output is required to measure a learning rate. Such a definition is obvious for manufacturing firms (cf. Nembhard and Uzumeri, 2000). For a software development organization, Boh et al. (2007) defined completed modification requests and software releases as units of output, which were accumulated over a time span of 14 years.

qualification to head a project team. Then, a requirement r_{pst} for this special skill s could model the tasks that must be accomplished by the project manager of project p in period t.

Finally, we address the compatibility between workers. Compatibility between workers means how well two workers cooperate. Compatibility is a complex matter, as it is affected by various personality traits and the work situation (cf. Tett and Murphy, 2002).

Findings about the relationship between worker compatibility and team performance are ambiguous. Intuitively, one would expect compatibility to be positively correlated to performance, as found by Reddy and Byrnes (1972) in an experimental study. However, Hill (1975) found in an empirical study of teams within an IT department that rather incompatibility than compatibility is associated with performance and effectiveness. Hill (1975) suggested that the nature of a task may impact the relation between worker compatibility and effectiveness: The more cooperation the task requires, the more important is compatibility. Though, if "synergistic gains are not possible, incompatibility may lead to higher total accomplishment through the channeling of energy into individual efforts" (Hill, 1975, p. 218).

Our models do not take care of worker compatibility. We assume that workers are equally compatible to one another, i.e., that they are indifferent to the selection of their co-workers.

In the literature, models have been proposed that consider worker compatibility. For example, Kumar et al. (2013) have formulated a MIP model for a problem where tasks must be assigned to workers and where some tasks depend on other tasks. If task j depends on task i, the workers that are assigned to these tasks must cooperate. The more the workers are compatible with each other, the better is their cooperation and the smoother is the corresponding work flow. The objective of the model of Kumar et al. (2013) is to assign tasks to workers such that the total pairwise compatibility of workers who must cooperate is maximized.

We could adopt the approach of Kumar et al. (2013) to modeling compatibility and we could integrate compatibility into our models in two ways. First, we could integrate compatibility into the objective function by adding a term that measures total pairwise compatibility of an assignment of workers to project teams. Second, we could integrate compatibility into the constraints of our models if we wish that total pairwise compatibility within project teams does not fall below a certain level. Both ways imply terms that are nonlinear in the decision variables x_{kp} or y_{kpst}, respectively. Alternatively, additional binary variables would allow to stick to a linear model.

Measuring total pairwise compatibility of a solution to the assignment problem can be done more or less detailed. We could merely consider the x_{kp} variables to measure compatibility solely based on project team membership. Alternatively, we could consider the y_{kpst} variables to weight the compatibility of workers based on actual cooperation. If, for example, worker k contributes to project p only in period $t = 1$, and worker k' contributes to project p only in period $t = 2$, the compatibility between k and k' will not matter much. Although worker k and worker k' join the same project team, the y_{kpst} variables reveal that the need for face-to-face interaction between k and k' is presumably small.

Considering compatibility would require a large quantity of personal data. It might be difficult to obtain these data and to obtain correct data, because workers would have to

reveal the quality of their working relationship to colleagues and are likely to give biased judgements.

If we only want to avoid that two workers k and k' are assigned to the same project, because they are likely to impede project work due to interpersonal conflicts, Constraint set (4.33) can be added to our models. Constraints (4.33) ensure for each project $p \in \mathcal{P}_k^{\text{suit}} \cap \mathcal{P}_{k'}^{\text{suit}}$ that at most one worker of the pair (k, k') is assigned to project p.

$$x_{kp} + x_{k'p} \leq 1 \qquad p \in \mathcal{P}_k^{\text{suit}} \cap \mathcal{P}_{k'}^{\text{suit}} \tag{4.33}$$

If, on the other hand, worker k and worker k' are an inseparable team and neither of them can work without the other, Constraints (4.34) can be added to our models. For each project for which both k and k' are suitable, Constraints (4.34) guarantee that either both are assigned to this project or neither of them.

$$x_{kp} = x_{k'p} \qquad p \in \mathcal{P}_k^{\text{suit}} \cap \mathcal{P}_{k'}^{\text{suit}} \tag{4.34}$$

4.4 Two alternative models for the utilization leveling problem

At the last stage of our three-stage hierarchical planning approach, we want to level the hours worked for the members of each department $d \in \mathcal{D}$ in each period $t \in \mathcal{T}$. The hours worked by employee k in period t comprise the time that he spends for projects and the time that he devotes to his department. At the last stage, the time that worker k will spend for projects in each period $t \in \mathcal{T}$ is already known and given by $\sum_{p \in \mathcal{P}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst}$. The time yd_{kt} that worker k must work for his department in a period t is yet to be determined such that the hours worked by worker k and his colleagues are leveled in this period.

In leveling problems, loads must be determined such that the loads are as equal as possible. Various objective functions for leveling problems have been proposed and considered in the literature, e.g., (1) minimizing the sum of squared loads (cf. Burgess and Killebrew, 1962), (2) minimizing the weighted sum of underloads and overloads (cf. Shanker and Tzen, 1985), (3) minimizing the maximum load (cf. Berrada and Stecke, 1986), (4) minimizing the absolute deviations between desired loads and planned loads (cf. Easa, 1989), (5) minimizing the difference between maximum and minimum load (cf. Guerrero et al., 1999), and (6) minimizing the total pairwise difference of loads or the average pairwise difference of loads (cf. Jang et al., 1996; and Kumar and Shanker, 2001, respectively). Objective functions (1), (2), (4) and (6) are suitable for our problem. We will consider two typical objective functions: a quadratic one, which follows (1), and a linear one, which follows (6).

A quadratic objective function for leveling problems sums the squares of the loads that are to be leveled. In our case loads are hours worked. The sum of loads has to be minimized. Loads that are disproportionately large are punished in the objective function by squaring.

Example 4.3 Assume that department d has two workers k_1 and k_2 whose project contributions in period t are 0. Let $rd_{dt} = 4$ and let $yd_{k_1t}^2 + yd_{k_2t}^2$ be the objective function,

which must be minimized. Then $yd_{k_1t} = yd_{k_2t} = 2$ is the optimal solution with an objective function value of 8. An allocation with $yd_{k_1t} = 1$ and $yd_{k_2t} = 3$, which is less balanced, would result in an objective function value of 10. □

For a quadratic objective function, our problem of allocating departmental workload is given by (4.35)–(4.38). The aim of model (4.35)–(4.38) is to level the total workload of workers who belong to department d in period t. This model has to be solved for each department $d \in \mathcal{D}$ in each period $t \in \mathcal{T}$, i.e., it has to be solved $D \cdot T$ times. The decision variables of the quadratic leveling model are the non-negative continuous variables yd_{kt}, $k \in \mathcal{K}_d$, which represent the departmental workload that is allocated to worker k in period t.

$$\text{Min.} \quad \sum_{k \in \mathcal{K}_d} \left(yd_{kt} + \sum_{p \in \mathcal{P}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst} \right)^2 \tag{4.35}$$

$$\text{s. t.} \quad \sum_{k \in \mathcal{K}_d} yd_{kt} = rd_{dt} \tag{4.36}$$

$$yd_{kt} \leq R_{kt} - \sum_{p \in \mathcal{P}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst} \qquad k \in \mathcal{K}_d \tag{4.37}$$

$$yd_{kt} \geq 0 \qquad k \in \mathcal{K}_d \tag{4.38}$$

Objective function (4.35) minimizes the sum of squared working times of workers from department d in period t. Constraints (4.36) ensure that the entire departmental workload is distributed among department members. Constraints (4.37) guarantee that the time that worker k spends for his department and for projects does not exceed his availability R_{kt}. Constraints (4.38) state the domains of the decision variables.

Model (4.35)–(4.38) can be transformed into an LP by linearizing the quadratic objective function (cf. Williams, 1999, pp. 136–142). Objective function (4.35) is separable, because it can be stated as a sum of terms dependent on a single variable. Each quadratic term yd_{kt}^2, $k \in \mathcal{K}_d$, of the objective function can be approximated by a piecewise linear function. Introducing such a piecewise linear function to the model requires additional variables to represent the line segments of the piecewise linear function: For n line segments, $n + 1$ continuous variables are required; for each line segment, two variables correspond to the endpoints of the interval for which the line segment approximates the quadratic function. Since objective function (4.35) is convex and we minimize this function, the sketched way of linearization, which leads to a linear program, works (cf. Williams, 1999, pp. 139-140). This means that the linearized model matches every feasible value of a variable yd_{kt} with the correct point on the correct line segment.

An optimal solution to the linearized model can deviate from an optimal solution to the original quadratic model. In general, the finer the approximation, i.e., the more line segments are used to describe a quadratic function, the closer the solution of the LP comes to the solution to the original problem.[5]

[5] If all load variables yd_{kt}, $k \in \mathcal{K}_d$, were restricted to integer values, an exact linearization would be possible, though, it would result in a MIP (cf. Rieck et al., 2012).

For our problem, it is also possible to apply a linear objective function, which minimizes the total absolute difference between the working times of all pairs (k, k') of workers, $k \in \mathcal{K}_d$, $k' \in \mathcal{K}_d \setminus \{k\}$. The absolute difference between working times of two workers k and k' can be calculated by the absolute value function. The absolute value function is not a linear function but can be linearized. This linearization is exact, in contrast to the linearization described for the quadratic model. Before we turn to the linearization of the absolute value function, let us consider an example that shows how the absolute value function evaluates different allocations of departmental workload. Our example takes up Example 4.3.

Example 4.4 Assume that department d has two workers k_1 and k_2 whose project contributions in period t are 0. Let $rd_{dt} = 4$ and let $\left|yd_{k_1t} - yd_{k_2t}\right|$ be the objective function, which must be minimized. Then $yd_{k_1t} = yd_{k_2t} = 2$ is the optimal solution with an objective function value of 0. An allocation with $yd_{k_1t} = 1$ and $yd_{k_2t} = 3$, which is less balanced, would result in an objective function value of 2. □

Let us consider the absolute value function from Example 4.4 to demonstrate how an absolute value function can be linearized. For linearization of the function $|yd_{kt} - yd_{k't}|$, we have to introduce a variable $\Delta_{kk'} \in \mathbb{R}_{\geq 0}$. If we require $\Delta_{kk'} \geq yd_{kt} - yd_{k't}$ and $\Delta_{kk'} \geq yd_{k't} - yd_{kt}$, then minimizing $\Delta_{kk'}$ is equivalent to minimizing $|yd_{kt} - yd_{k't}|$.

Now, the linear leveling model can be formulated. It is given by model (4.39)–(4.44). The aim of model (4.39)–(4.44) is to level the total workload of workers who belong to department d in period t. This model has to be solved for each department $d \in \mathcal{D}$ in each period $t \in \mathcal{T}$, i.e., it has to be solved $D \cdot T$ times. The decision variables are the non-negative continuous variables yd_{kt}, $k \in \mathcal{K}_d$, $t \in \mathcal{T}$, which represent the departmental workload that is allocated to worker k in period t. Auxiliary variables are the non-negative continuous variables $\Delta_{kk'}$, $k \in \mathcal{K}_d \setminus \left\{k_{|\mathcal{K}_d|}\right\}$, $k' \in \mathcal{K}_d$, $k' > k$, which represent the absolute difference between the hours worked by workers k and k'. Here, $k_{|\mathcal{K}_d|}$ denotes that worker of department d whose index k is the largest among all department members.

$$\text{Min.} \quad \sum_{k \in \mathcal{K}_d \setminus \left\{k_{|\mathcal{K}_d|}\right\}} \; \sum_{k' \in \mathcal{K}_d \,|\, k' > k} \Delta_{kk'} \tag{4.39}$$

$$\text{s. t.} \quad \Delta_{kk'} \geq yd_{kt} + \sum_{p \in \mathcal{P}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst} - \left(yd_{k't} + \sum_{p \in \mathcal{P}_{k'}^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{k'p}^{\text{match}}} y_{k'pst} \right) \quad \begin{array}{l} k \in \mathcal{K}_d \setminus \left\{k_{|\mathcal{K}_d|}\right\}, \\ k' \in \mathcal{K}_d,\ k' > k \end{array} \tag{4.40}$$

$$\Delta_{kk'} \geq yd_{k't} + \sum_{p \in \mathcal{P}_{k'}^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{k'p}^{\text{match}}} y_{k'pst} - \left(yd_{kt} + \sum_{p \in \mathcal{P}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst} \right) \quad \begin{array}{l} k \in \mathcal{K}_d \setminus \left\{k_{|\mathcal{K}_d|}\right\}, \\ k' \in \mathcal{K}_d,\ k' > k \end{array} \tag{4.41}$$

$$\sum_{k \in \mathcal{K}_d} yd_{kt} = rd_{dt} \tag{4.42}$$

$$yd_{kt} \leq R_{kt} - \sum_{p \in \mathcal{P}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst} \qquad k \in \mathcal{K}_d \tag{4.43}$$

$$yd_{kt} \geq 0 \qquad k \in \mathcal{K}_d \tag{4.44}$$

Objective function (4.39) minimizes the sum of pairwise absolute differences between working times of members of department d in period t. Constraints (4.40) assure that the difference in working times is registered for each pair (k, k'), $k > k'$, where worker k works more hours than worker k'. Constraints (4.41) register the difference in working times for each pair (k, k'), $k > k'$, where worker k works less than worker k'. Constraints (4.42) ensure that the entire departmental workload is distributed among department members. Constraints (4.43) guarantee that the time worker k spends for his department and for projects does not exceed his availability R_{kt}. Finally, Constraints (4.44) state the domains of the decision variables yd_{kt}. The domains of the auxiliary variables $\Delta_{kk'}$ are implicitly defined in Constraints (4.40) and (4.41).

An optimal solution for the quadratic model is also optimal for the linear model and vice versa. Hence, both models are equivalent. Though, from a computational point of view, the linear model seems preferable.

4.5 A monolithic model for all three problems

After we presented models for each stage of the hierarchical planning approach, we will show an integrated, monolithic model for our three problems. The monolithic model can serve as a reference point, which enables us to assess the efficiency of the hierarchical planning approach.

In Section 4.1 we outlined two roads to integrate multiple objectives into a monolithic model. The first road was to consider a weighted sum of the single objectives; the alternative was to optimize a vector of objective functions, where each single objective constituted a component of the vector. The monolithic model that we present here features an objective function that is a weighted sum of the objectives of our three problems.

Let w_1, w_2 and w_3 denote the weights of our three objectives. Weight w_1 corresponds to the objective of selecting the most beneficial project portfolio. Weight w_2 refers to the goal of minimizing the number of assignments of workers to selected projects. Finally, factor w_3 weights the impact of the aim to level working times of workers.

Note that the weight ratios w_1/w_2 and w_2/w_3 must be carefully chosen to obtain desired and sensible results. If the ratio w_1/w_2 is too low, the number of assignments might be minimized by selecting no project at all. In the integrated model, working times cannot only be leveled by allocating departmental workload, but also by allocating project workload. If the ratio w_2/w_3 is too low, the working time could be balanced optimally by allocating project workload to many workers leading to a high number of assignments. If the weight ratios are sufficiently high, the three objectives are lexicographically ordered as in the hierarchical approach.

The monolithic model is given by (4.45)–(4.57). Model (4.45)–(4.57) is a MIP model that comprises two types of binary decision variables and three types of non-negative continuous variables. Binary decision variables are the variables z_p, $p \in \tilde{\mathcal{P}}$, which indicate whether project p is selected or not, and the variables x_{kp}, $k \in \mathcal{K}$, $p \in \hat{\mathcal{P}}_k^{\text{suit}}$, which

indicate whether worker k is assigned to project p or not. The first type of non-negative continuous decision variables are the variables $\hat{y}_{kpst}$, $k \in \mathcal{K}$, $p \in \hat{\mathcal{P}}_k^{\text{suit}}$, $s \in \mathcal{S}_{kp}^{\text{match}}$, $t \in \mathcal{T}_p$, which represent the workload that worker k performs for project p and skill s in period t. The second type of non-negative continuous decision variables are the variables yd_{kt}, $k \in \mathcal{K}$, $t \in \mathcal{T}$, which represent the departmental workload that is allocated to worker k in period t. The third and last type of the non-negative continuous decision variables are the auxiliary variables $\Delta_{kk't}$, $k \in \mathcal{K}_d \setminus \left\{k_{|\mathcal{K}_d|}\right\}$, $k' \in \mathcal{K}_d$, $k' > k$, $d \in \mathcal{D}$, $t \in \mathcal{T}$, which represent the absolute difference in working times in period t between workers k and k' who belong to the same department d. Again, $k_{|\mathcal{K}_d|}$ denotes that worker of department $d \in \mathcal{D}$ whose index k is the largest among all members of department d.

With respect to the decision variables, there are three differences compared to the hierarchical models: First, one set of variables obtains an additional index. All variables used in the hierarchical models are also used in the monolithic model except for the variables $\Delta_{kk't}$. Because leveling cannot be done any longer for each period $t \in \mathcal{T}$ separately, we must add a time index to the variables $\Delta_{kk'}$, resulting in the variables $\Delta_{kk't}$. Both variables have the same meaning. They represent the absolute difference between the hours worked by workers k and k' in the considered period t. The second difference is that the number of the variables x_{kp} increased, because the number of projects that come into question for staffing increased from $|\mathcal{P}|$ to $\left|\mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}}\right|$. Finally, the variables $\hat{y}_{kpst}$ do not state the preliminary, but the final allocation of project workload.

$$\text{Min.} \quad -w_1 \sum_{p \in \tilde{\mathcal{P}}} b_p z_p + w_2 \sum_{k \in \mathcal{K}} \sum_{p \in \hat{\mathcal{P}}_k^{\text{suit}}} x_{kp} + w_3 \sum_{d \in \mathcal{D}} \sum_{k \in \mathcal{K}_d \setminus \left\{k_{|\mathcal{K}_d|}\right\}} \sum_{k' \in \mathcal{K}_d \,|\, k' > k} \sum_{t \in \mathcal{T}} \Delta_{kk't} \tag{4.45}$$

$$\text{s. t.} \quad \sum_{k \in \mathcal{K}_s} (l_{ks} \hat{y}_{kpst}) = r_{pst} z_p \qquad \begin{array}{l} p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}}, \\ s \in \mathcal{S}_p,\ t \in \mathcal{T}_p \end{array} \tag{4.46}$$

$$\sum_{s \in \mathcal{S}_{kp}^{\text{match}}} \hat{y}_{kpst} \leq R_{kt} x_{kp} \qquad \begin{array}{l} p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}}, \\ k \in \mathcal{K}_p^{\text{suit}} \setminus \mathcal{K}_p^{\text{assigned}},\ t \in \mathcal{T}_p \end{array} \tag{4.47}$$

$$\Delta_{kk't} \geq yd_{kt} + \sum_{p \in \hat{\mathcal{P}}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} \hat{y}_{kpst} - \left(yd_{k't} + \sum_{p \in \hat{\mathcal{P}}_{k'}^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{k'p}^{\text{match}}} \hat{y}_{k'pst} \right) \qquad \begin{array}{l} k \in \mathcal{K}_d \setminus \left\{k_{|\mathcal{K}_d|}\right\},\ k' \in \mathcal{K}_d, \\ k' > k,\ d \in \mathcal{D},\ t \in \mathcal{T} \end{array} \tag{4.48}$$

$$\Delta_{kk't} \geq yd_{k't} + \sum_{p \in \hat{\mathcal{P}}_{k'}^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{k'p}^{\text{match}}} \hat{y}_{k'pst} - \left(yd_{kt} + \sum_{p \in \hat{\mathcal{P}}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} \hat{y}_{kpst} \right) \qquad \begin{array}{l} k \in \mathcal{K}_d \setminus \left\{k_{|\mathcal{K}_d|}\right\},\ k' \in \mathcal{K}_d, \\ k' > k,\ d \in \mathcal{D},\ t \in \mathcal{T} \end{array} \tag{4.49}$$

$$\sum_{k \in \mathcal{K}_d} yd_{kt} = rd_{dt} \qquad d \in \mathcal{D},\ t \in \mathcal{T} \tag{4.50}$$

$$yd_{kt} + \sum_{p \in \hat{\mathcal{P}}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} \hat{y}_{kpst} \leq R_{kt} \qquad k \in \mathcal{K}_d,\, t \in \mathcal{T} \tag{4.51}$$

$$z_p = 1 \qquad p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \tag{4.52}$$

$$x_{kp} = 1 \qquad p \in \mathcal{P}^{\text{ongoing}}, k \in \mathcal{K}_p^{\text{assigned}} \tag{4.53}$$

$$z_p \in \{0,1\} \qquad p \in \tilde{\mathcal{P}} \tag{4.54}$$

$$x_{kp} \in \{0,1\} \qquad p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}},\; k \in \mathcal{K}_p^{\text{suit}} \setminus \mathcal{K}_p^{\text{assigned}} \tag{4.55}$$

$$\hat{y}_{kpst} \geq 0 \qquad k \in \mathcal{K},\, p \in \hat{\mathcal{P}}_k^{\text{suit}},\; s \in \mathcal{S}_{kp}^{\text{match}},\, t \in \mathcal{T}_p \tag{4.56}$$

$$yd_{kt} \geq 0 \qquad k \in \mathcal{K},\, t \in \mathcal{T} \tag{4.57}$$

Objective function (4.45) minimizes the weighted sum of our three objectives. Note that maximizing the weighted portfolio benefit is achieved by multiplying the weighted portfolio benefit by -1 and by minimizing the resulting term.

Constraint set (4.46) ensures that each requirement r_{pst} of project p is satisfied if project p is selected. Constraints (4.47) link the variables x_{kp} and $\hat{y}_{kpst}$. These constraints guarantee that worker k can only contribute to project p if he is assigned to project p. A contribution $\hat{y}_{kpst} > 0$ for any skill $s \in \mathcal{S}_{kp}^{\text{match}}$ in any period $t \in \mathcal{T}_p$ requires $x_{kp} = 1$. Simultaneously, Constraints (4.47) force $x_{kp} = 1$ if worker k contributes to project p for any skill $s \in \mathcal{S}_{kp}^{\text{match}}$ in any period $t \in \mathcal{T}_p$. Hence, if worker k contributes to project p, he is automatically assigned to project p.

Constraints (4.48) and (4.49) assure that the absolute difference in working times is registered for each pair (k, k') of workers within each department d in each period t. Constraints (4.50) ensure that the entire departmental workload is distributed among department members for each department d in each period t. Constraints (4.51) guarantee that the time that worker k spends for his department and for projects in period t does not exceed his availability R_{kt}.

Constraint set (4.52) fixes the variables z_p, $p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}}$, to 1, because we have to include these projects in the portfolio. For each ongoing project $p \in \mathcal{P}^{\text{ongoing}}$, a team $\mathcal{K}_p^{\text{assigned}}$ of workers exists already. For each member k of this team, Constraint set (4.53) fixes the corresponding variable x_{kp} to 1. Eventually, Constraint sets (4.54)–(4.57) state the domains of the actual decision variables. The domains of the auxiliary variables $\Delta_{kk't}$, which are required to obtain a linear term for the leveling goal in the objective function, are implicitly defined in Constraints (4.48) and (4.49).

4.6 Complexity analysis

In this section, we will draw upon complexity theory and use its findings and methods to judge whether our three problems are computationally easy or hard to solve. If we can show that a problem can be classified as a hard problem according to complexity theory, there is almost no hope to find an exact algorithm that solves any instance of the problem to optimality in adequate time. Then, heuristic solution methods can be a resort. Hence, the results of this section can guide our search for solution methods. We

will first introduce the key concepts of complexity theory in Subsection 4.6.1, before we analyze each of our three problems in the following Subsections 4.6.2–4.6.4 separately. The results are summarized in Subsection 4.6.5.

4.6.1 Basic concepts of complexity theory

In this subsection, we will give an overview of basic concepts of complexity theory. Complexity theory deals with—among other things—the complexity of algorithms and the complexity of problems. Algorithms can be classified according to their running time. We will distinguish polynomial-time from exponential-time algorithms. Problems can be assigned to different complexity classes according to their hardness. We will explain the most common complexity classes and sketch how membership of a problem in these classes can be proved.

Complexity theory comprises two main branches (cf. Garey and Johnson, 1979; Wegener, 2003). The first branch addresses running times and memory requirements of algorithms. The running time of an algorithm is also called its time complexity. The second branch entails the hardness of problems. The hardness of problems is also called its complexity. As we will see, both branches are related. The roots of complexity theory lie in the areas of computer science, mathematics, and operations research and started to flourish in the late 1960s (cf. Ahuja et al., 1993, p. 788).

It was said that the first branch of complexity theory addresses running times and memory requirements of algorithms. Nowadays, often the running time of an algorithm is in the spotlight, while memory requirements of an algorithm are less important, because memory space has become abundant and cheap. Since fast solution processes can save money and since algorithms can be enormously complex, the running time of an algorithm remains an important issue, even though processor speed has drastically increased, while processor prices have not increased in the recent decades.

With respect to running times of algorithms, complexity theory seeks to determine the minimum, average, and maximum running time that is required by an algorithm to solve any instance of a problem for which the algorithm was developed. For meaningful statements, the time required is expressed in relation to the instance size (cf. Garey and Johnson, 1979, p. 5). To compare different algorithms for the same problem, usually the maximum time required is considered (cf. Ahuja et al., 1993, pp. 56–57; Wegener, 2003, pp. 25–27; Nemhauser and Wolsey, 1999, p. 119), i.e., comparisons are based on worst-case time complexity.

The maximum time that an algorithm requires to solve instances of a given size is represented by an upper bound O on the number of elementary computational operations that are executed by the algorithm to solve such an instance. The upper bound O is an asymptotic upper bound, which only holds when instance size approaches infinity (Nemhauser and Wolsey, 1999, p. 119; Wegener, 2003, pp. 25–27, cf.). It is assumed that every elementary operation takes one unit of time (cf. Wegener, 2003, p. 22; Schirmer, 1995, p. 3). The number of elementary operations is expressed as a function f of the instance size or, in other words, as a function of the amount of information necessary to represent the instance. The instance size is given by one or more parameters. For our problem, the number of workers K and the number of projects P are parameters that impact instance size.

If only one parameter n specifies the size of an instance, Ahuja et al. (1993, p. 59) define that "an algorithm is said to run in $O(f(n))$ time if for some numbers c and n_0, the time taken by the algorithm is at most $cf(n)$ for all $n \geq n_0$". As an example, consider the time complexities $O(n^2)$, $O(2^n)$ and $O(n!)$. For $O(n^2)$ the function $f(n)$ is a polynomial in n. An algorithm whose running time is bounded by a polynomial in the instance size is called *polynomial-time algorithm*. If for some algorithm the function $f(n)$ is not a polynomial in n, as it is the case with $O(2^n)$ and $O(n!)$, the algorithm is said to run in exponential time and is termed *exponential-time algorithm* (cf. Garey and Johnson, 1979, p. 6).

If more than one parameter describes the size of an instance, the definitions of the previous paragraph apply analogously. As an example, let K and P define the instance size of a problem. Running times of $O(K + P^2)$ or $O(KP)$ are called polynomial, whereas running times of $O(K^P)$ or $O(KP!)$ are called exponential.

The second branch of complexity theory addresses the hardness of problems and classifies problems as computationally easy or hard to solve by assigning them to different complexity classes. Before we outline the most important complexity classes for our work, we will briefly distinguish *decision problems* from *optimization problems* and consider what their difference implies for complexity analysis, because the division of problems into different complexity classes is primarily done for decision problems, whereas our three problems are optimization problems.

Decision problems are problems whose solution is either "yes" or "no" (cf. Garey and Johnson, 1979, p. 18). For example, let us consider an instance of the problem of allocating project workload to workers. The following questions state decision problems: Does a feasible solution exist for the given instance? Does a feasible solution with an objective function value of 5 or less exist for the given instance?

We are, however, concerned with optimization problems, which belong to the broader class of search problems. We want to answer questions such as the following one: "Which feasible allocation of project workload for a given instance requires the least number of assignments of workers to projects?"

Before we explain how the hardness of a decision problem can be related to the hardness of an optimization problem, we define two phrases that are used in the following:

(1) an optimization problem and its *corresponding* decision problem, and

(2) a decision problem and its *corresponding* optimization problem.

Let us define the phrases by examples in a rather informal way. For (1) assume that the optimization problem "Max. (Min.) b subject to some constraints" is given. Then, the corresponding decision problem is defined as "Is there a feasible solution for the optimization problem with $b \geq c$ ($b \leq c$)?". Vice versa, for (2) consider the following decision problem: "Is there a solution with $b \geq c$ ($b \leq c$) that observes all constraints of a given problem?" Then, the corresponding optimization problem is defined as "Max. (Min.) b subject to all constraints of the given problem". Based on our definition, the corresponding optimization problem of a decision problem is uniquely defined, while the corresponding decision problem of an optimization problem is uniquely defined except for the value of c. Since the value of c is not relevant for our purposes, we consider the corresponding decision problem of an optimization problem as uniquely defined.

Garey and Johnson (1979, pp. 109–117) and Wegener (2003, pp. 50–53) use the concept of Turing reducibility, also called polynomial reducibility (cf. Schirmer, 1995, pp. 18–20), to show how results for the complexity of a decision problem can be transferred to the corresponding optimization problem. Later, we will consider this concept in more detail. For now, we content ourselves with the fact that the concept of Turing reducibility allows to conclude that an optimization problem is hard if its corresponding decision problem is hard. Hence, we can examine the decision problems that correspond to our optimization problems in order to obtain insights into the complexity of the optimization problems.

A focus of complexity theory is on the class NP (*non-deterministic polynomial-time*), which contains all decision problems that can be solved by a non-deterministic algorithm in polynomial time. Such an algorithm is a theoretical type of algorithm that decomposes the process of solving a problem into two stages: a *guessing stage* and a *checking stage*. At the first stage, a solution to the problem is guessed. The solution is also called instance. This is the non-deterministic guessing stage. At the second stage, it is checked in polynomial time if the solution is feasible (yes-instance) or not (no-instance). This is the polynomial-time checking stage. The statement that a problem that is in NP can be solved in polynomial time by this theoretical type of algorithm refers only to the checking stage and only to the case where a yes-instance is checked (cf. Nemhauser and Wolsey, 1999, pp. 128–129). The decision problems that correspond to our three optimization problems belong to the class NP what we will prove for each problem in the following subsections.

A major contribution of complexity theory is the classification of decision problems that belong to the class NP into three different subclasses. These three main subclasses are the class P (*deterministic polynomial-time*) of "easy" problems, the class NPC (*NP-complete*) of "hard" problems, and the class NPI (*NP-intermediate*) of problems whose complexity is between the complexities of the classes P and NPC (cf. Garey and Johnson, 1979, pp. 154–161). Problems in class P can be solved by polynomial-time algorithms. For problems of class NPC only exponential-time algorithms are known, but until now it could not be ruled out that polynomial-time algorithms for these problems might be developed. If a polynomial-time algorithm was found for one problem out of NPC, every problem of the class NPC could be solved in polynomial time. This would imply $\mathsf{P} = \mathsf{NP}$. Though, a majority of researchers supposes that the conjecture $\mathsf{P} \neq \mathsf{NP}$ is true. A proof or a falsification of this conjecture, however, remains to be presented and is currently the presumably greatest challenge in the field of complexity theory.

Among the problems that belong to the class NPC, there is a special type of problems called *number problems* (cf. Garey and Johnson, 1979, pp. 90–106). Some of these number problems are computationally better tractable than other number problems and non-number problems. We will succinctly treat this special type of problems in order to be able to draw potential conclusions for our problems.

Number problems are problems where the largest number that appears within the instance data cannot be bounded by a polynomial in the instance size. As an example for a number problem, consider the knapsack problem, where a subset from n items has to be selected that has maximum total utility and a total weight not exceeding the knapsack capacity c. An instance comprises n weights, n utility values and the capacity value c. Without loss of generality, let c be the largest number of the instance. The instance size, i.e., the amount of information necessary to represent the instance is bounded by

$O(n \log_2 c)$ if numbers are encoded using the binary representation. Because the maximum number c cannot be bounded by $O(n \log_2 c)$, the knapsack problem is a number problem.

Another number problem, which also belongs to NPC, is the traveling salesman problem. In this problem, we seek a tour of minimum length through n cities. The distance between any two cities cannot be bounded by a polynomial in the instance size.

In contrast, the minimum cover problem[6], which also belongs to NPC, is not a number problem, i.e., it is a non-number problem. In the minimum cover problem, a set C is given consisting of sets C_i, $i = 1, \ldots, |C|$, where each set C_i is a subset of a finite set F. Additionally, a positive integer $b \leq |C|$ is given. The question is whether C contains a cover of F that has a size of at most b, i.e., whether there is a subset $C' \subseteq C$ with $|C'| \leq b$ such that every element of F is contained in at least one set $C_i \in C'$ (cf. Garey and Johnson, 1979, p. 222). The only and thus largest number that appears in an instance of the minimum cover problem is the integer b. The instance size is bounded by $O(|C||F| + \log_2 b)$. Since $b \leq |C|$ holds by definition, b is bounded by a polynomial in C and hence by a polynomial in the instance size. Consequently, the minimum cover problem is not a number problem.

As for all NP-complete problems, no algorithms for number problems have been found whose running time is bounded by a polynomial in the instance size. However, for some number problems, algorithms exist whose running time is bounded by a polynomial in the instance size and in the maximum number of the instance. These algorithms are called *pseudopolynomial-time algorithms*. Those number problems that can be solved by pseudopolynomial-time algorithms are better tractable than all other problems in NPC, be it number problems or not.

As an example for a number problem that can be solved in pseudopolynomial time, consider the knapsack problem. It can be solved by dynamic programming in $O(nc)$ time by filling in a table with nc cells (cf. Garey and Johnson, 1979, pp. 90–92; Martello and Toth, 1990, p. 7). This time complexity implies that instances of the knapsack problem can be solved in polynomial time with respect to instance size if the largest number c of these instances is rather small such that c can be bounded by a polynomial in the number of items n.

On the other side, there are number problems for which no pseudopolynomial-time algorithm is known, e.g., the traveling salesman problem. Even if all numbers within an instance of the traveling salesman problem are small, it takes exponential time to solve the instance. Even if all intercity distances are either 1 or 2, for example, no polynomial-time algorithm is known.

Due to the existence of these two types of number problems, the class NPC is further divided. All problems that belong to the class NPC are called NP-complete. Number problems for which no pseudopolynomial-time algorithm is known and all non-number problems in NPC are called NP-*complete in the strong sense* and belong to the subclass sNPC (*strongly* NP-*complete*) (cf. Garey and Johnson, 1979, p. 95). For instance, the traveling salesman problem and the minimum cover problem are strongly NP-complete, whereas the knapsack problem is only NP-complete.

The distinction of number problems from non-number problems is relevant for our work, because arbitrarily large numbers can appear in instances of our three problems. Consider the workforce assignment problem, for example. For the decision and opti-

[6]The minimum cover problem is also called set-covering problem.

mization version of this problem, instance size is bounded by $O(KT\log_2(\max_{k,t} R_{kt}) + KS\log_2(\max_{k,s} l_{ks}) + DT\log_2(\max_{d,t} rd_{dt}) + PST\log_2(\max_{p,s,t} r_{pst}))$. Thus, the worker availabilities R_{kt}, the departmental requirements rd_{dt}, and the project requirements r_{pst} are not bounded by a polynomial in the instance size. Recall that we bounded the skill levels l_{ks} by 2.

In the following three subsections we will prove that the decision problems that correspond to our three optimization problems belong to P, NPC or even sNPC. To prove membership in the class P for a decision problem Π, it is sufficient to state a polynomial-time algorithm which solves Π. For a detailed description of proving techniques with regard to membership in the classes NPC and sNPC see Garey and Johnson (1979, pp. 63–74 and 95–106). We will only sketch how membership in these two classes is proved by giving two formal proof definitions. Additionally, we briefly explain a proving technique called *restriction*, which facilitates the application of the rather theoretical and formal proof definitions.

To prove that a problem Π belongs to the class NPC, two steps are necessary. We first have to show that Π belongs to NP. Finally, a problem Π^{NPC} that is known to be in the class NPC must be polynomially transformed to Π such that a yes-instance (no-instance) of Π^{NPC} is transformed into a yes-instance (no-instance) of Π. If such a polynomial transformation exists and if an algorithm existed that would solve Π, this algorithm would also solve Π^{NPC}. Since no polynomial-time algorithm exists for Π^{NPC} if $\mathsf{P} \neq \mathsf{NP}$ is true, there exists no polynomial-time algorithm for Π and thus Π must belong to NPC.

To prove that a number problem Π^{Num} belongs to the class sNPC, two alternative ways are possible. Both ways are closely related to the way just outlined for proving NP-completeness. The first alternative requires two steps. First, we have to restrict problem Π^{Num} to a problem Π_p^{Num} where all numbers are bounded by a polynomial p in the instance size. Second, we must show that this problem Π_p^{Num} is NP-complete. The second alternative requires to show that Π^{Num} belongs to the class NP and that a problem Π^{sNPC} that is known to be strongly NP-complete can be pseudopolynomially transformed to Π^{Num}.[7]

To apply the proof definitions, which essentially rely on a transformation of one problem into another, proving techniques have emerged. A common technique for proving that a problem Π is (strongly) NP-complete uses the principle of *restriction* (cf. Garey and Johnson, 1979, pp. 63–66). If a (strongly) NP-complete problem exhibits a one-to-one correspondence to the problem Π, it can be easily transformed to Π. Though, often it is difficult to find such a (strongly) NP-complete problem. Then, it might be possible to restrict Π to a special case for which a transformation from a known (strongly) NP-complete problem can easily be constructed. Restricting Π to a special case means that Π is restricted to a proper subset of all its instances. If it can be shown that the restricted problem is (strongly) NP-complete, so its generalization Π is, because there is no algorithm that solves *any* instance of Π in (pseudo)polynomial time. We will apply the principle of restriction in the following subsections.

The classes NP, P, NPC, and sNPC refer to decision problems only, but for optimization problems a comparable classification is common. If a decision problem is NP-complete, the corresponding optimization problem is called NP-*hard*, and if a decision problem is

[7] For the definition of a pseudopolynomial transformation see Garey and Johnson (1979, pp. 101–106).

strongly NP-complete, the corresponding optimization problem is called NP-*hard in the strong sense* or just *strongly* NP-*hard* (cf. Schirmer, 1995, p. 20 and p. 26). The term (strongly) NP-hard means that a (strongly) NP-hard optimization problem is at least as hard as any decision problem that is (strongly) NP-complete.

To show that a (strongly) NP-hard optimization problem is at least as hard as any (strongly) NP-complete decision problem, the aforementioned concept of Turing reducibility can be applied. A Turing reduction of a problem Π to a problem Π' is similar to a polynomial transformation except for the fact that a Turing reduction allows to solve Π by repeatedly calling a subroutine to solve different instances of Π' (cf. Garey and Johnson, 1979, p. 111; Schirmer, 1995, pp. 118–120). An optimization problem Π_{opt} is (strongly) NP-hard if there is an NP-complete problem Π^{NPC} (strongly NP-complete problem Π^{sNPC}) that Turing-reduces to Π_{opt}. Since any decision problem Turing-reduces to its corresponding optimization problem, an optimization problem is (strongly) NP-hard if the corresponding decision problem is (strongly) NP-complete. Hence, we can derive complexity results for our optimization problems from the complexity results that we obtain for their corresponding decision problems.

4.6.2 Complexity of the project selection problem

In this subsection, we prove that our project selection problem is NP-hard in the strong sense.

The project selection problem was described in Section 3.2 and modeled in Section 4.2. The corresponding decision problem asks whether there is a feasible portfolio with a total benefit of at least b, $b \in \mathbb{N} \setminus \{0\}$.

To simplify our presentation, we abbreviate our optimization problem as $\text{MPSWS}_{\text{opt}}$ (multi-project skilled workforce selection problem) and its corresponding decision problem as $\text{MPSWS}_{\text{dec}}$. Additionally, let the vector $\boldsymbol{z}$ represent values for all variables z_p, $p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}}$, and let the matrix $\hat{\boldsymbol{y}}$ represent values for all variables $\hat{y}_{kpst}$, $k \in \mathcal{K}$, $p \in \hat{\mathcal{P}}_k^{\text{suit}}$, $s \in \mathcal{S}_{kp}^{\text{match}}$, $t \in \mathcal{T}_p$.

Lemma 4.1 $\text{MPSWS}_{\text{dec}} \in \text{NP}$. □

Proof Let $(\boldsymbol{z}, \hat{\boldsymbol{y}})$ be a (guessed) solution for an arbitrary instance of $\text{MPSWS}_{\text{dec}}$. We can check in polynomial time if this solution is feasible and if the portfolio that is associated with $\boldsymbol{z}$ has a total benefit of at least b. The feasibility check requires that we test whether the solution satisfies Constraint sets (4.2)–(4.7). The number of constraints within these sets is bounded by a polynomial in the instance size. ■

Theorem 4.1 $\text{MPSWS}_{\text{dec}}$ *is strongly* NP-*complete.* □

Proof (Polynomial transformation from Minimum Cover)
Problem: Minimum Cover
Instance: Set C containing sets C_i, $i = 1, \ldots, |C|$, where each set C_i is a subset of a finite set F; positive integer $b \leq |C|$.
Question: Does C contain a cover of F that has a size of at most b, i.e., is there a subset $C' \subseteq C$ with $|C'| \leq b$ such that every element of F is contained in at least one set $C_i \in C'$?

MINIMUM COVER is NP-complete in the strong sense even if $|C_i| \leq 3$ holds for all $C_i \in C$ with $|C_i| = 3$ for at least one C_i (cf. Garey and Johnson, 1979, p. 222).

For this proof, we restrict $\text{MPSWS}_{\text{dec}}$ to the case where $T = 1$; $R_{kt} = 1$, $k \in \mathcal{K}$, $t = 1$; $rd_{dt} = 0$, $d \in \mathcal{D}$, $t = 1$; $\left|\mathcal{P}^{\text{ongoing}}\right| = \left|\mathcal{P}^{\text{must}}\right| = 0$ and $\left|\tilde{\mathcal{P}}\right| = K+1$, i.e., where $K+1$ projects can be selected.

We assume that the set of skills $\mathcal{S}$ is a union of two disjoint sets $\mathcal{S}^F$ and $\mathcal{S}^{\text{unique}}$ with $\left|\mathcal{S}^{\text{unique}}\right| = K$. Each unique skill $s \in \mathcal{S}^{\text{unique}}$ is mastered by exactly one worker and each worker $k \in \mathcal{K}$ masters one unique skill $s \in \mathcal{S}^{\text{unique}}$ and between one and three additional skills from $\mathcal{S}^F$. For each skill $s \in \mathcal{S}$ and each worker $k \in \mathcal{K}$, we presume $l_{ks} = 1$.

From the set $\tilde{\mathcal{P}}$ of projects, K projects are assumed to require only one skill, namely, a unique skill $s \in \mathcal{S}^{\text{unique}}$, but no two of these projects p, $p = 1, \ldots, K$, require the same unique skill, i.e., each unique skill $s \in \mathcal{S}^{\text{unique}}$ is required by exactly one project. For each project p, $p = 1, \ldots, K$, its requirement is given by $r_{pst} = 1$, $s \in \mathcal{S}_p$, $t = 1$, and the benefit is given by $b_p = 1$. The remaining project $p = K + 1$ requires all the skills $s \in \mathcal{S}^F$, but no skill from $\mathcal{S}^{\text{unique}}$. Let project $p = K + 1$ have requirements $0 < r_{pst} \leq \frac{1}{3}$, $s \in \mathcal{S}^F$, $t = 1$, and a benefit $b_p = K+1$. Note that our restriction of $\text{MPSWS}_{\text{dec}}$ leads to problem instances in which all numbers are bounded by a polynomial in the instance size.

To tie an instance of MINIMUM COVER to an instance of $\text{MPSWS}_{\text{dec}}$, we associate the set F with the set $\mathcal{S}^F$ of skills that are required by project $p = K + 1$. Let each subset C_i be associated with a worker k and let the elements in C_i correspond to the skills in $\mathcal{S}_k \cap \mathcal{S}^F$ that are mastered by the associated worker apart from his unique skill. Then there exists a cover of F that has size b or less if and only if a feasible portfolio can be selected with a total benefit of at least $K + 1 + (K - b)$.

Note that a feasible portfolio with a total benefit of $K + 1 + (K - b)$ necessitates that $K - b$ workers must spend their entire time to accomplish the project that requires their unique skill $s \in \mathcal{S}^{\text{unique}}$. This necessity leaves only b workers who can contribute to project $p = K + 1$. Together with Lemma 4.1 our proof is complete. ■

Corollary 4.1 $\text{MPSWS}_{\text{opt}}$ *is strongly* NP*-hard.* □

To be more precise, the proof has shown that $\text{MPSWS}_{\text{opt}}$ is strongly NP-hard if every worker masters at least one unique skill and if at least one worker masters four skills or more in total. By a second proof, we will show that not only instances where each worker masters a unique skill and some other skills are strongly NP-hard but also instances where each worker masters only one skill. In addition, the second proof points to a special case that can be solved in pseudopolynomial time. Our second proof for Theorem 4.1 reads as follows.

PROOF (Pseudopolynomial transformation from MULTIDIMENSIONAL KNAPSACK)
Problem: MULTIDIMENSIONAL KNAPSACK (cf. Kellerer et al., 2004, pp. 235–238)
Instance: Set J of n items; set I of m dimensions such as weight, volume and concentration; benefit p_j, $j \in J$; size w_{ij}, $i \in I$, $j \in J$, of item j with respect to dimension i; positive integer c_i, $i \in I$, that represents the capacity with respect to dimension i; positive integer b.
Question: Is there a subset $J' \subseteq J$ such that $\sum_{j \in J'} w_{ij} \leq c_i$, $i \in I$, and $\sum_{j \in J'} p_j \geq b$?

MULTIDIMENSIONAL KNAPSACK is NP-complete, because it comprises the NP-complete (one-dimensional) knapsack problem as a special case ($m = 1$). MULTIDIMENSIONAL KNAPSACK can be solved by dynamic programming in $O\left(n \left(\max_i c_i\right)^m\right)$ time

by filling a table with $n \times c_1 \times c_2 \times \cdots \times c_m$ cells (cf. Kellerer et al., 2004, pp. 248–252). Hence, if the number of dimensions m is bounded by a constant, MULTIDIMENSIONAL KNAPSACK can be solved in pseudopolynomial time, but in general, MULTIDIMENSIONAL KNAPSACK is strongly NP-complete (cf. Kaparis and Letchford, 2008, p. 91).

To prove NP-completeness of $\text{MPSWS}_{\text{dec}}$ by transformation from MULTIDIMENSIONAL KNAPSACK, we restrict $\text{MPSWS}_{\text{dec}}$ to the special case where $\mathcal{P}^{\text{ongoing}} = \mathcal{P}^{\text{must}} = \emptyset$; $T = 1$; $rd_{dt} = 0$, $d \in D$, $t = 1$; $\mathcal{S}_p = \mathcal{S}$, $p \in \tilde{\mathcal{P}}$; and $|\mathcal{S}_k| = 1$, $k \in \mathcal{K}$, i.e., where each worker masters only one skill. Additionally, we assume $l_{ks} = 1$, $k \in \mathcal{K}$, $s \in \mathcal{S}_k$.

Let each project $p \in \tilde{\mathcal{P}}$ be associated with an item j and let each skill s out of the finite set of skills be associated with a dimension i. Let for all projects $p \in \tilde{\mathcal{P}}$ each requirement r_{pst}, $s \in \mathcal{S}_p$, $t = 1$, be associated with the corresponding item size w_{ij} and let the knapsack capacities c_i, $i \in I$, correspond to the total workforce availability with regard to the associated skill s given by $\sum_{k \in \mathcal{K}_s} R_{kt}$, $t = 1$. Then, an instance of MULTIDIMENSIONAL KNAPSACK is a yes-instance if and only if the associated instance of $\text{MPSWS}_{\text{dec}}$ is a yes-instance. Together with Lemma 4.1 our proof is complete. ∎

Corollary 4.2 $\text{MPSWS}_{\text{opt}}$ *is strongly* NP*-hard even if each worker masters only one out of several skills. The special case of* $\text{MPSWS}_{\text{opt}}$ *that corresponds to the restricted decision problem outlined in the second proof of Theorem 4.1 can be solved in pseudopolynomial time for any fixed number of skills* $|\mathcal{S}|$. □

However, this special case of $\text{MPSWS}_{\text{opt}}$ where only mono-skilled workers with homogeneous skill levels are considered is far off those practical cases that we are interested in. The project selection problem that prevails in practice is NP-hard in the strong sense.

4.6.3 Complexity of the workforce assignment problem

In this subsection, we show that our optimization problem of assigning workers to projects and allocating project workload to workers is NP-hard in the strong sense.

The problem was described in Section 3.3 and modeled in Subsection 4.3.1. The corresponding decision problem asks whether there is a feasible allocation of project workload that results in no more than b assignments of workers to projects, $b \in \mathbb{N} \setminus \{0\}$.

To shorten our presentation, we abbreviate our optimization problem as $\text{MPSWA}_{\text{opt}}$ (multi-project skilled workforce assignment problem) and its corresponding decision problem as $\text{MPSWA}_{\text{dec}}$. Additionally, let the matrix $\boldsymbol{x}$ represent values for all variables x_{kp}, $k \in \mathcal{K}$, $p \in \mathcal{P}_k^{\text{suit}}$, and let the matrix $\boldsymbol{y}$ represent values for all variables y_{kpst}, $k \in \mathcal{K}$, $p \in \mathcal{P}_k^{\text{suit}}$, $s \in \mathcal{S}_{kp}^{\text{match}}$, $t \in \mathcal{T}_p$.

Lemma 4.2 $\text{MPSWA}_{\text{dec}} \in$ NP. □

PROOF Let $(\boldsymbol{x}, \boldsymbol{y})$ be a (guessed) solution for an arbitrary instance of $\text{MPSWA}_{\text{dec}}$. We can check in polynomial time if the solution is feasible and if the number of assignments of workers to projects that is associated with $\boldsymbol{x}$ does not exceed b. A check whether the solution is feasible requires to test if the solution satisfies Constraint sets (4.10)–(4.16). The number of constraints within these sets is bounded by a polynomial in the instance size. ∎

Theorem 4.2 $\text{MPSWA}_{\text{dec}}$ *is strongly* NP*-complete.* □

Proof (Polynomial transformation from Minimum Cover)
We restrict $\text{MPSWA}_{\text{dec}}$ to the special case where $P = 1$; $\mathcal{S}_p = \mathcal{S}$, $p = 1$; $T = 1$; $rd_{dt} = 0$, $d \in \mathcal{D}$, $t = 1$. Let us assume that $R_{kt} = \infty$, $k \in \mathcal{K}$, $t = 1$, i.e., that the workers' availabilities are unbounded.[8]

Now, let the set F of an instance of Minimum Cover (see page 79) be associated with the set $\mathcal{S}$ of skills. Let each subset C_i be associated with a worker k and let the elements of C_i correspond to the skills in $\mathcal{S}_k$ that are mastered by the associated worker. Then there is a cover of F that has size b or less if and only if there is a feasible allocation of the workload of project $p = 1$ with a team size of b or less. Together with Lemma 4.2 our proof is complete. ∎

Remark 4.1 If the unbounded version of the decision problem is strongly NP-complete, all the more is the bounded version. □

Remark 4.2 Minimum Cover can be solved in polynomial time by matching techniques if $|C_i| \leq 2$ holds for all $C_i \in C$. We will succinctly outline an efficient solution procedure that incorporates matching techniques, but first we will give some definitions on which the procedure is founded.

An undirected graph $G(N, E)$ is a set N of nodes and a set E of edges. Each edge $e \in E$ joins two nodes u and v. Both nodes u and v are said to be incident with e. A matching M in graph G is a subset of edges in E such that every node $v \in N$ is incident with at most one edge in M (cf. Burkard et al., 2009, p. 2; Jungnickel, 2005, p. 205). A matching M is a maximum matching if its cardinality $|M|$ is maximal, i.e., if there is no matching M' with $|M'| > |M|$.

To tackle an instance of Minimum Cover by a matching technique, we construct a graph G by establishing a node v for each element $f \in F$ that is given in the instance of Minimum Cover. For each subset $C_i \in C$ with $|C_i| = 2$, an edge e is established that joins those two nodes u and v that correspond to the elements $f \in C_i$.

To find a cover of F, a maximum matching M is determined for this graph G, e.g., by the algorithm of Edmonds (cf. Edmonds, 1965; Jungnickel, 2005, pp. 374–396). If the number of nodes $|N|$ is even and $|M| = \frac{|N|}{2}$, then the matching M represents a cover of F. Otherwise, M represents only a partial cover. If M represents only a partial cover of F, we extend this partial cover step by step, until all elements of F are covered. In each step, we either extend the partial cover by a subset C_i with $|C_i| = 2$ whose corresponding edge does not belong to M or by a subset C_i with $|C_i| = 1$ whose element f has not been covered yet.

The minimum number of subsets C_i necessary for a cover of F is given by the sum of the cardinality $|M|$ of the matching and the number of subsets C_i that are required to extend the partial cover to a full cover. This minimum number can be compared to b to answer the question whether there is a cover of size b or less. □

[8] Instead of unbounded worker availabilities we could alternatively choose the project requirements r_{pst} for the sole project $p = 1$ so small that every worker k could accomplish the workload for all skills $s \in \mathcal{S}_{kp}^{\text{match}}$ of his matching skills if he was assigned to the project. In consequence, the largest number that appears in an instance of the restricted version of $\text{MPSWA}_{\text{dec}}$ is bounded by a polynomial in the instance size.

Remark 4.2 is relevant for instances of our assignment problem where no worker masters more than two skills and where the project requirements are very small when compared to the availabilities of workers. For these instances, the solution procedure that was outlined in Remark 4.2 can be applied to find the minimum team size if only one project must be staffed. If more projects must be staffed, say $P \geq 2$ projects, then the solution procedure must be applied P times, once for each project.

For a clearer picture of the complexity of $\text{MPSWA}_{\text{dec}}$ with respect to its subproblems, we will provide a second proof showing that $\text{MPSWA}_{\text{dec}}$ is strongly NP-complete. Our second proof is not redundant in so far, as it shows that there are even instances of $\text{MPSWA}_{\text{dec}}$ that feature only one skill but cannot be solved in polynomial or pseudopolynomial time. Our first proof did only show that instances with $|\mathcal{S}_k| \geq 3$ for at least one worker k are intractable. The following second proof, which relies on transformation from 3-Partition, will reveal conditions where instances with $|\mathcal{S}| = 1$ are intractable. Our second proof does not render our first proof redundant, as the first proof shows that $\text{MPSWA}_{\text{dec}}$ is strongly NP-complete even if worker availabilities are unbounded. Hence, both proofs shed precious light on the frontier between hard and easy problems (cf. Garey and Johnson, 1979, pp. 80–90).

As a vehicle for the second proof, we use the network model, which was introduced in Subsection 4.3.1, to represent our assignment problem. Before we will present the proof, let us shortly turn towards two variants of a network design problem that are related to our assignment problem and that led us the way to our proof. The two variants are the *fixed-charge network flow problem* and the *minimum edge-cost flow problem*.

Both the fixed-charge network flow problem (cf. Kim and Pardalos, 1999; Cruz et al., 1998; Magnanti and Wong, 1984) and the minimum edge-cost flow problem (cf. Garey and Johnson, 1979, p. 214) seek for a minimum cost origin-destination flow on a network with bounded arc capacities where fixed arc costs are incurred whenever an arc transports a positive flow.

There are only slight differences between the two network design problems. The fixed-charge network flow problem considers not only fixed arc costs, but in addition also variable arc costs which linearly depend on the amount of flow shipped on the arc. Furthermore the flow variables are continuous variables, whereas the minimum edge-cost flow problem presumes integral flow variables.

Both network design problems are NP-complete in the strong sense. For the fixed-charge network flow problem, Guisewite and Pardalos (1990) prove this problem complexity by transformation from 3-Satisfiability (abbreviated 3SAT, cf. Garey and Johnson, 1979, p. 259). For the minimum edge-cost flow problem, strong NP-completeness is indicated by Garey and Johnson (1979, p. 214), who refer to a transformation from Exact Cover by 3-Sets (abbreviated X3C, cf. Garey and Johnson, 1979, p. 221). This transformation from Exact Cover by 3-Sets is shown by Benoist and Chauvet (2001, pp. 2–3), for example.

To recognize the relation of our assignment problem to network design, see that we could restrict our problem to a special case of the fixed-charge network flow problem. For our proof, however, we will exploit the relation to the minimum edge-cost flow problem. Our proof follows Benoist and Chauvet (2001, pp. 5–6), who show that the minimum edge-cost flow problem remains strongly NP-complete for a special case which they call

bipartite minimum edge-cost flow problem.[9] Their proof relies on transformation from 3-PARTITION. We will take on this transformation to prove that $\text{MPSWA}_{\text{dec}}$ is intractable even if it features only one skill.

Theorem 4.3 $\text{MPSWA}_{\text{dec}}$ *is strongly* NP*-complete even if restricted to one skill, i.e., even if* $|\mathcal{S}| = 1$. □

PROOF (Pseudopolynomial transformation from 3-PARTITION)
Problem: 3-PARTITION
Instance: Set C that contains $3m$ elements, $m \geq 3$, $m \in \mathbb{N}$; a positive integer B; a size $s(c) \in \mathbb{N} \setminus \{0\}$ for each $c \in C$ such that $\frac{B}{4} < s(c) < \frac{B}{2}$ for all $c \in C$ and $\sum_{c \in C} s(c) = mB$.
Question: Can C be partitioned in m disjoint subsets $C_1, C_2, \ldots, C_m$ such that $\sum_{c \in C_i} s(c) = B$ holds for $i = 1, \ldots, m$? (If so, every subset C_i must contain exactly three elements from C.)

3-PARTITION is NP-complete in the strong sense (cf. Garey and Johnson, 1979, p. 224).

To prepare a transformation, we picture an instance of 3-PARTITION by a bipartite graph $G(N, A)$ with node set N and arc set A. Graph G is depicted in Figure 4.2. The set N of nodes is a union of the disjoint node sets U and V. Arcs run only from nodes in U to nodes in V and arc capacity is not bounded. The graph G represents a directed network with a set of supply nodes, namely, the set U, and a set of demand nodes, namely, the set V. Concretely, for each element $c \in C$ a node u is established whose supply is equal to the size $s(c)$. The node set V comprises m nodes, which have an identical supply of $-B$, i.e., a demand of B units of flow. The arc set comprises the arcs $\langle u, v \rangle$, $u \in U$, $v \in V$.

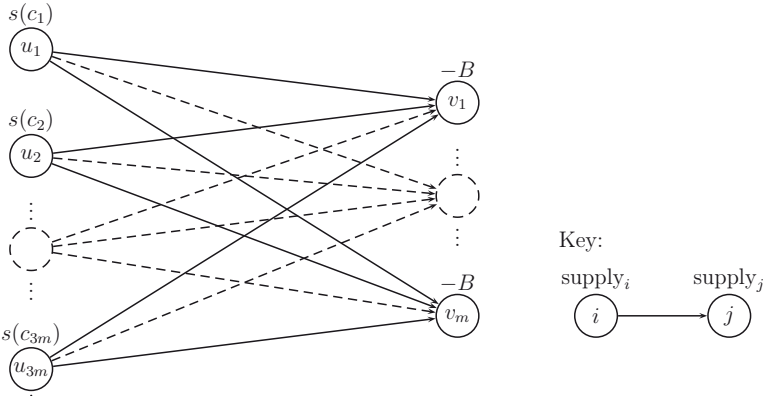


Figure 4.2: Network representation of an instance of 3-PARTITION for transformation to $\text{MPSWA}_{\text{dec}}$

[9] The bipartite minimum edge-cost flow problem was formulated by Benoist and Chauvet (2001) for a problem in construction industry. This problem is described in detail by Benoist (2007).

A solution to an instance of 3-PARTITION is represented by a flow from nodes $u \in U$ to nodes $v \in V$ that does not exceed the supply of nodes in U and that satisfies all demands of nodes in V and that does not require more than $3m$ arcs to transport the flow units.

For our proof, we restrict $\text{MPSWA}_{\text{dec}}$ to the special case where $T = 1$; $D = 1$; $P = m - 1$; $S = 1$; and $l_{ks} = 1$, $k \in \mathcal{K}$, $s = 1$. In this case, all workers belong to the same unique department and all projects require the same unique skill.

For a transformation from 3-PARTITION to $\text{MPSWA}_{\text{dec}}$, let each element c, i.e., each node u, correspond to a worker k and let the supply of node u, i.e., $s(c)$, be associated with the availability R_{kt}, $t = 1$, of the corresponding worker k. Furthermore, we associate each of the $m - 1$ projects with a demand node v and we associate the requirement r_{pst} of a project p for the unique skill s in period $t = 1$ with the demand of the corresponding node v. Finally, let the sole department $d = 1$ correspond to a node v and let its requirement r_{dt}, $t = 1$, correspond to the demand of the node v. This demand amounts to B units of flow.

Then, an instance of 3-PARTITION is a yes-instance if and only if there is an allocation of project workload such that the number of assignments of workers to projects does not exceed $3(m - 1)$. Note that $3(m - 1)$ assignments imply that each project is staffed with three workers, who spend their entire available time for this project, while three workers who are not assigned to any project spend their entire time to accomplish the departmental workload. Together with Lemma 4.2 our proof is complete. ∎

Remark 4.3 Since 3-PARTITION remains strongly NP-complete as long as $m \geq 3$, $\text{MPSWA}_{\text{dec}}$ is NP-complete in the strong sense if two projects must be staffed and a unique department has a positive requirement, or if at least three projects must be staffed. □

Remark 4.4 The special case where two projects must be staffed and no departmental workload arises, is NP-complete. This complexity result can be concluded from equivalent proofs of Guisewite and Pardalos (1990) and Benoist and Chauvet (2001). Both contributions show by transformation from SUBSET SUM (cf. Garey and Johnson, 1979, p. 223) that the corresponding fixed-charge network flow problem and the corresponding minimum edge-cost flow problem, respectively, are NP-complete if there are only two demand nodes. By their proofs, both works help to clarify the sight on the boundary line between NP-completeness and strong NP-completeness for the respective problem.

Note that the conclusion for our assignment problem does not hold if only one project must be staffed and a positive requirement of one department must be satisfied. Although this case results in two demand nodes, transformation from SUBSET SUM is not possible in this case. □

To sum up, we have the following three complexity results for the optimization version $\text{MPSWA}_{\text{opt}}$ of our assignment problem.

Corollary 4.3 $\text{MPSWA}_{\text{opt}}$ *can be solved in polynomial time if workers master at most two skills and worker availabilities are "unbounded".* □

Corollary 4.4 $\text{MPSWA}_{\text{opt}}$ *is* NP*-hard if two projects must be staffed and worker availabilities are bounded.* □

Corollary 4.5 $\text{MPSWA}_{\text{opt}}$ *is* NP*-hard in the strong sense if at least one worker masters three or more skills that are required by a project, or if at least three projects must be staffed and worker availabilities are bounded, or if two projects must be staffed and in the sole department some work must be accomplished and worker availabilities are bounded.* □

4.6.4 Complexity of the utilization leveling problem

In this subsection, we state that the utilization leveling problem is solvable in polynomial time.

Our optimization problem of allocating workload of a department such that the working times of workers who belong to the department are leveled as well as possible was described in Section 3.4 and modeled in Section 4.4. The corresponding decision problem asks whether there is a feasible allocation of departmental workload such that the sum of pairwise absolute differences in working times is not greater than b, $b \in \mathbb{N} \setminus \{0\}$.

Theorem 4.4 *The decision problem of our leveling problem is in* P*.* □

Proof The decision problem of our leveling problem can be solved in polynomial time. A polynomial-time algorithm that solves the optimization problem in $O(K^2)$ time is presented in Section 6.1.2. ■

Corollary 4.6 *The optimization version of our leveling problem can be solved in polynomial time.* □

4.6.5 Summary of results

Table 4.1 summarizes the main results of the complexity analysis that we conducted for our three problems.

Table 4.1: Main results of the complexity analysis for the three problems considered in this thesis

Problem	Complexity
Project selection	NP-hard in the strong sense
Workforce assignment	NP-hard in the strong sense
Utilization leveling	Solvable in polynomial time

Chapter 5

Literature review

In this chapter, we review the literature that is related to the three problems introduced in the previous two chapters. For the project selection and the workforce assignment problem, a broad overview of related work has already been given in Section 2.1. Here, we narrow our view on publications that have close links to our problems and we review solution methods for related problems in more detail. Close links to our problems exist, for example, when multi-skilled workers are considered or when objective functions take team size or workload leveling into account. In Section 5.1, work that is closely related to the project selection problem is reviewed. Section 5.2 gives an overview of contributions that deal with staffing problems where the number of required workers plays an important part. In Section 5.3, we look at articles that contain approaches to leveling worker utilization.

5.1 Work related to the project selection problem

A broad choice of literature related to static project selection problems has been presented and classified in Table 2.1 in Section 2.1. Here, we concentrate on methods for composing a portfolio of projects and on works that are closely related to our selection problem. First, we will give a brief overview of methods that have been used for portfolio selection. Then, we will consider two works in detail that explicitly take multi-skilled workers with heterogeneous skill levels into account.

Various quantitative approaches to project portfolio selection have been outlined by Martino (1995) and Graves and Ringuest (2003), for example. An overview has been given by Heidenberger and Stummer (1999). Although, Heidenberger and Stummer focus on the selection of research and development projects, their classification is applicable for a wide range of project selection problems. Our short presentation of methods follows their classification. Heidenberger and Stummer distinguish six classes of methods: (1) benefit measurement methods, (2) mathematical programming approaches, (3) methods from the realms of decision theory and game theory, (4) simulation models, (5) heuristic methods, and (6) cognitive emulation approaches. Selection procedures that are applied in practice often combine methods from several of these classes. In the following, we describe each class in more detail.

ad (1): Benefit measurement methods comprise comparative methods, scoring approaches, economic models, and group decision techniques. Comparative methods are based on numerous comparisons between projects, whereas projects are rated separately in scoring approaches. An example for a comparative method is the analytic hierarchy process (AHP), which is described by Saaty (1994). When

M. Walter, *Multi-Project Management with a Multi-Skilled Workforce,* Produktion und Logistik, DOI 10.1007/978-3-658-08036-5_5, © Springer Fachmedien Wiesbaden 2015

applied to project selection, the AHP uses pairwise comparisons between criteria to determine criteria weights and pairwise comparisons between projects with respect to each criterion in order to derive a ranking of projects. The AHP method was applied to project selection by Muralidhar et al. (1990), Lai and Xue (1999), and Tavana (2003), for example. In typical scoring approaches, decision makers agree on a set of selection criteria and a weight for each criterion. Then, they assess for each project and each criterion on a point scale how well the project meets the criterion. The weighted sum of all points that a project obtained represents the project score and allows a ranking of the projects. Economic models are based on cost-benefit analyses or on net present value calculations, for instance. An example for a group decision technique is the Delphi method.

ad (2): Mathematical programming approaches include linear, nonlinear, integer, and goal programming approaches among others. Integer programming has been used by Ghasemzadeh et al. (1999) and Kolisch et al. (2008), for example. Our model for project selection, which is a mixed-integer program, belongs also to the category of integer programming approaches. Among many others, Taylor et al. (1982) and Santhanam et al. (1989) have illustrated how goal programming can be applied to project selection.

ad (3): Examples for approaches from decision theory and game theory are decision-tree approaches and leader-follower models, respectively.

ad (4): Monte Carlo simulation is an example for a simulation technique that can be applied to project portfolio selection. This simulation technique allows, for instance, to depict a distribution of portfolio outcomes when project success is uncertain (cf. Souder and Mandakovic, 1986, p. 39; Martino, 1995, Chapter 6).

ad (5): Heuristic solution approaches are applied when instances of selection problems become too large and too demanding for exact methods. Many heuristic methods exist and have been applied to selection problems. Escudero and Salmeron (2005), for example, apply a relax-and-fix approach and Doerner et al. (2004) apply ant colony optimization, simulated annealing, and a genetic algorithm to a multiobjective selection problem.

ad (6): The last class covers cognitive emulation approaches. These approaches include statistical approaches among others. Statistical approaches are based on historical data. By regression analysis, for example, it is tried to identify crucial factors that affect project success.

Several of the aforementioned selection methods are applied in two works that are closely related to our project selection problem. The first of these two works is the work of Yoshimura et al. (2006). They consider a single-period project selection problem that integrates the assignment of workers to projects. Their problem setting is similar to ours, as workers belong to different departments and can form cross-departmental project teams. However, the departments themselves do not play a role in their approach, whereas we consider departmental workloads. In their approach, workers master different skills with levels of 0, 0.5, 1, and 2. Hence, they distinguish similar discrete levels as we do.

Yoshimura et al. outline a two-stage approach that applies a binary programming model at the first stage where candidate project portfolios are composed. At the second stage, a genetic algorithm is used for assigning workers to projects. The authors consider four objectives; one objective refers to the first stage, the others refer to the second stage. Their approach determines heuristically a set of efficient solutions with respect to the four objectives. From these efficient solutions, a decision maker can choose his favorite solution.

At the first stage, a binary program is formulated to determine not only one but several project portfolios. The objective function considers two aspects of a portfolio: the profit and the importance of the included projects. A weighted sum of portfolio profit and importance is maximized. Binary variables indicate whether a project is included in the portfolio or not. The binary program comprises only a single set of constraints. For each skill, it must be ensured that the workforce can satisfy the skill requirements of the portfolio. Though, the availabilities of workers are ignored at this stage. It is assumed that each worker contributes to the requirements of all matching skills of all projects. Consequently, it must only be guaranteed for each skill that the sum of the skill levels of all workforce members satisfies the skill requirement of the project portfolio. For the binary program, a set of different solutions, i.e., several project portfolios, are determined. Though, Yoshimura et al. do not outline how this is accomplished. When the first stage has been completed, each project portfolio that has been selected, is treated separately but in the same way.

In a step between the first and second stage, a project leader is determined for each project within a portfolio. A leader of a project must master the core skill of the project with a level of 2 and must be qualified for the job of a project leader. Yoshimura et al. do not indicate whether a worker can lead more than one project and whether a project leader must contribute to skill requirements of his project at all.

At the second stage, the problem of assigning workers to projects is formulated using variables $x_{kp} \in \{0, 0.1, 0.2, \ldots, 1\}$ where x_{kp} indicates for worker k the percentage of his total availability that he devotes to project p. The availabilities of all workers are identical. Since a worker cannot work overtime and can thus spend at most 100 % of his time for project work, he can contribute to at most ten projects. The size of a project team is not limited; theoretically, the complete workforce can be assigned to a project.

Three objectives are pursued at the second stage. The first is to satisfy skill requirements of projects as well as possible. In contrast to the first-stage constraint set, now the actual contribution x_{kp} of a worker k to project p is taken into account when the fulfillment of a skill requirement for project p is considered. The second objective is related to future career paths of the workers. A parameter specifies for each project p and each worker k how useful a contribution x_{kp} is for the development of worker k. The objective function that is associated with the second objective maximizes the total usefulness of assignments with respect to career considerations. The third objective regards workers' priorities for projects and compatibilities between members of each project team. The corresponding objective function integrates both aspects in a weighted sum that is maximized.

For the assignment problem at the second stage, solutions that are pareto-optimal with respect to the three second-stage objective functions are determined by a genetic algorithm. A solution for the variables x_{kp} that are associated with worker k is coded as a sequence of zeros and ones. Each zero represents a time budget of 10 %; the bit

sequence of each worker hence contains 10 zeros. The ones are used as separators between two adjacent projects. To give an example, consider a case where the selected portfolio contains three projects. Let the bit sequence of a worker be "000 1 0 1 000000". This sequence indicates that the worker devotes 30 % of his time to the first project, 10 % of his time to the second project, and the remaining 60 % to the third project. Other parts and procedures of the genetic algorithm remain untold; Yoshimura et al. (2006) outline neither how a crossover operation works or how mutation is done nor how parents are selected for recombination or how a new generation is built.

The efficient second-stage solutions that have been generated for a project portfolio are reassessed with respect to the first-stage objective. However, a refined variant of the first-stage objective function is used for the reassessment. The refined variant assumes that the assignment of workers influences project profits and hence takes the actual assignment of workers to projects into account.

Finally, the solutions of all portfolios that were selected at the first stage are considered. From these solutions, an efficient set with respect to the refined first-stage objective function and all three second-stage objective functions is determined. For the case where the resulting set is too large and confusing for a decision maker, reduction techniques are proposed.

Applying their approach to an instance of small size, Yoshimura et al. provide an illustrative example for their method. However, they do not present numerical results for larger-sized instances.

Compared to our approach, Yoshimura et al. (2006) model worker availabilities and project requirements less detailed. In their approach, all workers have identical availabilities and contributions of workers to projects are rather unspecified, as there is no proper allocation of workload. Here, the resolution of our selection model and our assignment model is more detailed. Furthermore, the matrix organization that Yoshimura et al. mention is not reflected in their approach, as the workload that arises in functional departments is ignored.

On the other hand, Yoshimura et al. take the effect of an assignment on project profits into account, whereas in our model the benefit b_p of a project p is independent from the project team and the workload allocation. Moreover, our approach does not consider career goals for workers, priorities of workers, and compatibilities between workers. These aspects are taken into account by Yoshimura et al. in the second and third objective function at the assignment stage. Considering these aspects implies, however, that a huge amount of data has to be collected before planning can start. The amount of data may be deterrent, but in practice the type of the required data, which belong to the category of personal information, may be even more problematic than their amount. In particular, it will be difficult to elicit reliable and undistorted information from workers about their compatibility with colleagues.

The second work that is closely related to our approach is the work of Gutjahr et al. (2008). They outline a multi-period problem that integrates project selection, scheduling, and staffing. In contrast to our approach, they model dynamic skills that vary over time due to learning and forgetting. Considering dynamic skills renders the objective function and some constraints of their formulation nonlinear. The objective of Gutjahr et al. is to determine a solution that maximizes project portfolio benefit and that results in a desired development of workers' skill levels.

Gutjahr et al. (2008) formulate a nonlinear mixed-integer program with various sets of decision variables. For each project, a benefit parameter is given and a binary variable indicates whether the project is selected or not. The projects are constituted of tasks. Each task j has a release date and a due date between which the respective workload has to be accomplished for each skill that is required by the task. The contribution of a worker k to a skill s required by task j in period t is denoted by a continuous variable x_{jkst}. The contributions of worker k in period t must observe his availability. The skill level or efficiency of worker k in skill s in period t is expressed by the variable $\gamma_{kst} \in [0, 1]$, whose value depends on the prior contributions of worker k to skill s. Since variables γ_{kst} are defined by terms that are nonlinear in the corresponding variables x_{jkst} and in t, a nonlinear model results. The objective function of the model is a weighted sum of portfolio benefit and skill improvements.

To solve their nonlinear problem, Gutjahr et al. (2008) outline two alternative heuristic solution approaches, one based on ant colony optimization, the other based on a genetic algorithm. The two meta-heuristics are used to generate project portfolios; both heuristics call the same staffing subprocedure for each portfolio in order to staff project tasks and to schedule the workload.

In the ant colony optimization approach, artificial ants construct project portfolios step by step. All projects are ordered in a sequence and each step of the portfolio construction process of an ant is associated with the project at the corresponding position in the sequence. In each step, an ant makes a stochastic decision whether to include the respective project in its current portfolio. The selection probability of a project depends on the pheromone value of the project. The pheromone values of projects are updated after each iteration when all ants have constructed their portfolio. For the pheromone updates, only the best portfolio of an iteration is relevant. Whenever an ant adds a project to its current portfolio and this additional project leads to an infeasible solution of the corresponding staffing subproblem, the ant just drops the project and continues the construction process with the next project in the sequence.

The genetic algorithm uses a binary string to represent a portfolio. Feasibility of a portfolio is assessed by the staffing subprocedure. If the subprocedure fails to find a (possibly existing) feasible staff assignment, a repair mechanism is invoked, which repeatedly removes a randomly chosen project from the portfolio until the staffing subprocedure returns a feasible allocation of workload. A string and a corresponding workload allocation constitute an individual. Fitness values of individuals are identical to the corresponding objective function values. For recombination, parent individuals are chosen by a roulette wheel selection. A one-point crossover is applied to their strings in order to create children. The children replace their parents in the population. Mutation is realized by a random bit flip and applied to each position in every newly created binary string.

The staffing subprocedure, which schedules and allocates workload, is a heuristic based on priority rules. This rather simple heuristic cannot guarantee to find an existing feasible solution for the variables x_{jkst} given a portfolio. A succinct description of the heuristic is provided on page 128 in Subsection 6.2.1, where we outline a related procedure for the workforce assignment problem. For the staffing subprocedure of Gutjahr et al. (2008), it remains unclear how and to which extent dynamic skills are considered.

Gutjahr et al. (2008) present additional constraint sets for their model in order to adapt the model to more realistic problem settings. One of these constraint sets limits

the maximum number of workers that can contribute to the skill requirements of a task. It requires additional binary variables. As motivation for this constraint set, they refer to the insight of Brooks (1982, p. 25) that increasing team size does not increase but decrease the probability of meeting a deadline.

Gutjahr et al. (2008) test their solution approaches for a real-life instance with 18 candidate projects, which comprise 20 tasks in total, 28 workers, 80 skills, and 24 periods and a slightly smaller-sized instance with 14 projects and 40 skills. For the smaller-sized instance without additional constraints, both solution approaches, the ant colony optimization algorithm and the genetic algorithm, exhibit a substantial gap compared to an exact solution of a linear approximation. Gutjahr et al. show that this gap can be attributed in a large part to the staffing subprocedure.

Compared to our approach with static skill levels, the resolution of skill levels is considerably higher in the model of Gutjahr et al. (2008). Their approach facilitates to pursue strategic goals with respect to the development of competencies. But again, additional input data are necessary to model learning and forgetting. Another difference between their and our approach is the extent of integration. We give priority to the project selection problem and solve it first. Then, we make or adjust staffing decisions in order to obtain small teams. Team size is a criterion that is less important than portfolio benefit in our view. Gutjahr et al., however, simultaneously decide about the portfolio and the assignment of workers to project tasks. They attach greater importance to staffing and skill development.

Gutjahr et al. (2008) also show how parts of their model and their complete model can be linearized. The linearizations lead to a quadratic and linear model, respectively. Though, they do not really exploit linearization but apply heuristic solution approaches instead. Only in Gutjahr et al. (2010), a follow-up paper, which treats an almost identical problem, linearization is actually deployed and comes into effect. In this paper, goals such as portfolio benefit and skill development are not integrated into a single objective function; instead, a vector of objective functions is considered where each objective function represents a single goal. Gutjahr et al. (2010) determine an approximation to the set of non-dominated solutions with respect to this vector of goals.

The selection procedures of Yoshimura et al. (2006) and Gutjahr et al. (2008) take multi-skilled workers with heterogeneous skills into account and compose a team of workers for each selected project. However, team size is of no particular concern in these works. In the following section, we review staffing approaches that explicitly consider team size.

5.2 Work related to the workforce assignment problem

In this section, we review the literature related to the workforce assignment problem. Note that a broad selection of related publications on scheduling and staffing has already been compiled and classified in Tables 2.2 and 2.3 in Section 2.1. Now, we take a closer look at approaches to scheduling and staffing where team size or workforce size is minimized. Our review is divided into two parts. In the first part, we consider problems that involve unrelated tasks or tasks that belong to a single project; in these problems, one main decision has to be taken for each worker, namely, the decision whether the worker is used or not. In the second part, we deal with multi-project problems where the important decision

about the use of a worker must be made for each project and where the assignment of a worker to one project can affect the team size of other projects. We begin the first part with a succinct review of the *fixed job scheduling problem* (FJSP) and some of its variants such as the *FJSP with machine availabilities*. In the FJSP, one seeks for the minimum number of workers or machines that is required to process a set of jobs whose start times have already been fixed. Next, we look at two works that tackle enriched variants of the FJSP and that are more closely related to the workforce assignment problem. Then, we turn to two works that focus on the effects of multi-skilling on workforce size and personnel costs, before we conclude the first part with the single-project problems of Li and Womer (2009a,b). At the beginning of the second part, we consider two articles that deal with single-period multi-project staffing problems and that limit the number of projects per worker and the number of workers per task via constraints. Then, we discuss the works of Grunow et al. (2004) and Heimerl and Kolisch (2010a) in greater detail because they are concerned with staffing multiple projects in a multi-period setting and are hence closely linked with our staffing approach. For a staffing subproblem in Grunow et al. (2004) which resembles the workforce assignment problem, we prove strong NP-hardness by reduction from FJSP WITH MACHINE AVAILABILITIES.

A well studied problem where the number of required resource units has to be minimized is the fixed job scheduling problem, which is also known as the *interval scheduling problem* (cf. Kolen et al., 2007). The FJSP can be described as follows. Given are identical machines that are always available and n jobs where each job $j = 1, \ldots, n$ has a start time $s_j \geq 0$ and a finish time $f_j > s_j$ and must be processed during the interval $[s_j, f_j)$ by a single machine. Each machine can process every job but at most one job at a time. The problem is to determine the minimum number of machines which is required to process all jobs and a corresponding assignment of jobs to machines. In such an assignment, which is also called a schedule, no two jobs whose intervals overlap must be assigned to the same machine.

The FJSP can be solved in polynomial time. Gupta et al. (1979) outline an algorithm that requires $O(n \log n)$ time; it sorts the $2n$ start and finish times associated with the n jobs in non-decreasing order and initializes a stack with n available machines. Then it scans the ordered list of start and finish times. When a start time is encountered, the corresponding job is assigned to the next available machine popped from the stack; when a finish time is encountered, the machine occupied by the corresponding job is released and pushed onto the stack. The initial sorting operation takes $O(n \log n)$ time, while the subsequent operations can be accomplished in $O(n)$ time.

When machines are non-identical, the corresponding fixed job scheduling problems become NP-hard, in general. We succinctly treat three basic variants of the FJSP with heterogeneous machines. The first variant is the *FJSP with machine availabilities*. Here, each machine i is associated with an interval or track $[a_i, b_i)$ during which it is available (cf. Kolen et al., 2007, section 4). The second variant is called *FJSP with given machines* or *k-track assignment problem*. In this variant, k machines that have individual availability periods or tracks as in the first variant are given and the problem is to determine a schedule with the greatest number of assigned jobs (cf. Brucker and Nordmann, 1994). Both variants are strongly NP-hard as Kolen et al. (2007, section 4.1) and Brucker and Nordmann (1994, p. 98) show by reduction from the problem of coloring circular arcs. That circular arc coloring is NP-complete has been proved by Garey et al. (1980).

The third variant arises when machines are identical with regard to availabilities but non-identical with regard to capabilities. In this case, each machine can process only an arbitrary subset of jobs. Again, an assignment of jobs to machines has to be found that uses a minimum number of machines. This variant, which we denote by *FJSP with machine capabilities*, is related to MINIMUM COVER; Kroon et al. (1997, section 3) prove by a reduction from 3-DIMENSIONAL MATCHING that this variant is strongly NP-hard even if preemption is allowed, i.e., even if it is allowed to interrupt the processing of a job on a machine and to continue the job on another machine.

The first and third variant, i.e., the FJSP with machine availabilities and the FJSP with machine capabilities, which both seek for the minimum number of required machines, are related to our workforce assignment problem. In the workforce assignment problem, the schedule of each project is fixed, just as the schedule in both variants of the FJSP. When we restrict the workforce assignment problem to the case where only one project has to be staffed, then minimizing the number of assignments of workers to the project is equivalent to minimizing the number of required workers. However, there are differences between the two variants of the FJSP on the one hand and the workforce assignment problem on the other hand. In the latter problem, the schedule of a project is not fixed within a period. A skill requirement can be accomplished any time within the corresponding period; a requirement does not have a defined start and finish time. Likewise, the availability of a worker in a period is not defined by a start and finish time but only by a time budget. With regard to capabilities of resources, there are also correspondences and differences. In both problems, a job or a skill requirement, respectively, can only be accomplished by a qualified resource. Yet, since skill levels are distinguished in the workforce assignment problem, the processing time of a skill requirement depends on the resource to which the requirement is assigned. Furthermore, a skill requirement can be assigned to several workers, i.e., preemption is allowed. Although there are significant differences, solution approaches to the FJSP with machine availabilities or to the FJSP with machine capabilities may provide helpful clues.

The second variant of the FJSP, the k-track assignment problem, which seeks for a job portfolio of maximum size, is related to the project selection problem tackled in this thesis. The given set of machines in the k-track assignment problem corresponds to the set of workers in the project selection problem and the jobs correspond to the projects. A most beneficial choice of jobs that can be processed by the machines can be associated with a feasible project portfolio of maximum benefit.

The works that we consider in the remainder of this section have been listed in Tables 2.2 and 2.3 in Section 2.1 but are considered in more detail now. We begin with two enriched variants of the FJSP.

Valls et al. (1996) consider a problem where the minimum number of workers has to be determined that is necessary to execute a machine load plan. A machine load plan is defined as a set of jobs with fixed start and finish times that have already been assigned to machines. The workforce is partitioned into worker classes. A subset of machines is associated with each worker class. The subset associated with a worker class indicates to which machines and thus to which jobs a worker of this class can be assigned. It is assumed that an unlimited number of workers is available from each worker class.

Valls et al. formulate the problem as a restricted vertex coloring problem. The formulation is based on a graph whose nodes represent the jobs and where an edge between

two nodes indicates that the intervals of the corresponding jobs overlap and that the jobs must not be assigned to the same worker. Each worker class is associated with a color c and each worker of a class is related to a unit of the color of his class. A node or vertex that corresponds to a job j must be colored by exactly one unit of a color c whose corresponding worker class contains job j in its set of assignable jobs. Furthermore, if any two adjacent nodes are colored by the same color c, they must be colored by different units of this color. The goal is to find a coloring that uses the smallest number of units of colors.

To solve this coloring problem, Valls et al. outline a branch-and-bound algorithm that applies problem-specific branching strategies, several upper and lower bounds, and dominance rules. They could solve instances with up to 15 worker classes, 300 jobs assigned to 15 machines, and on average 6 worker classes capable of operating a machine.

We adopted the ideas behind two lower bounds outlined by Valls et al. for lower bounds that we apply to the workforce assignment problem. We considered their lower bounds L^c and L_1 and derived a lower bound on the number of workers needed to accomplish the requirements of a project for a skill and a lower bound on the number of workers needed to staff a complete project. In Subsection 6.1.1 we introduce global and local versions of these bounds, which are termed LB_{ps}^{glob}, LB_{ps}^{loc}, LB_{p}^{glob}, and LB_{p}^{loc}. The lower bound L^c of Valls et al., which is related to LB_{ps}^{glob} and LB_{ps}^{loc}, determines for those nodes that can only be colored by a color c the minimum number of required units of color c. Analogously, we determine for each skill s that is needed by a project p the minimum number of workers needed by project p to cover its workload of skill s. The lower bound L_1, which is related to LB_{p}^{glob} and LB_{p}^{loc}, is a lower bound on the total number of units of colors needed to color all nodes in the graph considered by Valls et al. Analogously, we determine for each project p the minimum number of workers required to satisfy all skill requirements of project p.

Krishnamoorthy et al. (2012) tackle the FJSP with machine availabilities and capabilities for a workforce of limited size. For each worker that is required, fixed costs are incurred. In their mixed-integer programming formulation, Krishnamoorthy et al. minimize the sum of fixed costs over all required workers. For their computational study, however, they use identical cost values for all workers, so that they minimize the number of required workers.

To find a solution to their problem, Krishnamoorthy et al. have devised a heuristic approach that can be interpreted as a drop method. For a given workforce size, a two-stage procedure heuristically checks if a feasible assignment of jobs to workers exists. If the procedure finds an admissible assignment, a worker is removed or dropped from the workforce and the two-stage procedure is invoked again. Hence, the two-stage procedure can be regarded as the feasibility check for a drop operation. Note that we also outline a drop heuristic for our workforce assignment problem (see Subsection 6.2.3). In their paper, Krishnamoorthy et al. do not specify how the worker that is to be removed is chosen.

At the first stage of their two-stage procedure, the LP relaxation of the MIP model is considered and is further relaxed by a Lagrangian approach. For the resulting relaxation, an approximate solution is determined by a special subgradient algorithm. The solution provides Lagrangian multipliers, which are used at the second stage.

At the second stage, a Lagrangian relaxation of the original MIP is considered. Since the same constraint set as at the first stage is transferred into the objective function, the

Lagrangian multipliers determined at the first stage can be used as initial values for the multipliers at the second stage. The resulting Lagrangian relaxation decomposes into 1-track assignment problems, one for each worker. A 1-track assignment problem, which is a k-track assignment problem with $k = 1$, can be represented as a shortest path problem. These shortest path problems are solved by a label correcting algorithm. As long as the solutions, which are associated with single workers, do not constitute a feasible solution of the original MIP, the Lagrangian multipliers are adjusted by systematic perturbation and the 1-track assignment problems are solved again until a stopping criterion is met.

In a computational study, Krishnamoorthy et al. compared their approach to the exact MIP solver of the solver package CPLEX 11.2 for medium-sized instances and larger-sized instances with up to 2000 jobs and 420 workers capable of performing up to 66 % of all jobs. For medium-sized instances, CPLEX reached an average gap of 6.7 %, whereas the approach of Krishnamoorthy et al. comes off with an average gap of only 2.9 %. For those larger-sized instances for which CPLEX 11.2 failed to provide an admissible solution within half an hour, the heuristic approach provided feasible solutions with an average gap of 10.1 % to a lower bound provided by CPLEX.

Next, we consider the approaches of Brusco and Johns (1998) and Gomar et al. (2002), which minimize personnel costs that depend on workforce levels. Both works present MIP models for which solutions are determined by commercial solvers. The models are used to assess cross-training strategies; the corresponding results have been summarized in this thesis on pages 23–24 in Section 2.2, which deals with multi-skilling strategies.

The problem of Brusco and Johns (1998) features different worker classes. Workers that belong to the same class are identical in regard to skill sets and corresponding skill levels. For each skill and for each period of the planning horizon, a skill requirement is given that must be satisfied by workers. Costs are incurred for each worker that is deployed during the planning horizon; the costs can depend on the class of the worker. Moreover, a break period must be granted to each worker that is used. In the model, three decisions must be made. First, it must be decided how many workers from each worker class are required to accomplish all requirements such that costs are minimized. Second, for each period and for each skill, it must be determined how many workers of each class contribute to the respective skill requirement. Third, for each period and each worker class, the number of workers who rest in the period must be determined.

Brusco and Johns apply their model to a daily workload allocation problem of a U.S. paper mill, which involved 4 skills, 17 periods of half and hour length, and 4 worker classes comprising 200 workers in total. Since information on fixed costs for workers of the different classes was not available for the authors, they assumed identical costs across all worker classes and thus minimized the number of required workers.

Gomar et al. (2002) consider a problem that comprises short-term employment and assignment decisions for a construction project, which spans a few weeks. Given a set of mono- and multi-skilled workers, it must be decided for each day which workers are employed and which workers from those on the payroll are assigned to which skill requirement such that all skill requirements can be accomplished. A skill is associated with a craft and hence with a crew, e.g., with the crew of bricklayers, carpenters, or electricians. We term such a crew a *skill crew* or a *crew* for short.

The objective function defined by Gomar et al. integrates three components. These components are the sum of daily worker wages, the costs for hiring workers, and penalty

costs for cases where a worker is not assigned to the same skill crew as on the previous day. The first two components represent conflicting goals because they reflect the trade-off between costs incurred for idle resources on one side and costs for high turnover on the other side. The third component explicitly considers the detrimental effects of switching between skill crews and tries to reduce the variance in crew composition. However, switching between crews, which is enhanced through multi-skilling, can be beneficial when it reduces the number of new hires. To analyze multi-skilling strategies, an instance with 50 workers, 4 skills, and 16 days is considered.

The third component of the objective function in the model of Gomar et al. penalizes each switch of a worker between skills. Our model, in contrast, appreciates switching of a worker between skills within a project but penalizes switching between projects.

Li and Womer (2009a) consider a project scheduling problem that is associated with the mission of a naval ship. Shipboard operations, which must be executed by sailors, correspond to project tasks that must be scheduled within a deadline and subject to general temporal constraints. Each task requires a set of skills. Exactly one sailor must be assigned to each skill requirement of a task. Each sailor can be assigned to at most one skill requirement at a time and needs rest periods during the mission. The skill set of a sailor is not predefined but subject to optimization; at most four different skills can be assigned to a sailor. The problem is modeled as a MIP and the goal is to find a schedule that requires the minimum number of sailors.

Li and Womer (2009a) have outlined a three-phase decomposition approach to solve the problem heuristically. Their approach can be interpreted as an add method because the three-phase sequence is passed repeatedly, each time with an additional sailor. The first phase presumes that the number of available sailors is given. At the beginning, the number of available sailors is derived from a lower bound calculation. The first phase considers only the scheduling part of the problem and regards the sailors as homogeneous resources. A constraint programming approach is used to determine a feasible schedule. If no feasible schedule exists, the number of sailors is increased by one; sailors are added until a feasible schedule has been found.

In the second phase, the first-phase schedule is adjusted by a resource leveling procedure. A tabu search heuristic is applied in order to minimize the number of overlapping tasks. The resulting schedule constitutes the starting point of the third phase where a feasible assignment of skills to sailors and of sailors to skill requirements must be determined. This assignment problem is formulated as a special bin packing problem where each sailor corresponds to a bin and skill requirements and rest periods correspond to the items that must be packed into the bins. If no feasible solution is found, the number of sailors is increased by one and the first phase is invoked again. The three-phase sequence is repeated until a feasible solution to a third-phase assignment problem has been found.

In a computational study, Li and Womer (2009a) solved instances with up to 60 tasks, 8 skills, and 60 sailors that could be deployed. The decomposition approach found optimal or near-optimal solutions for smaller-sized instances. For larger-sized instances, it outperformed CPLEX 9.0, which was applied to the MIP formulation of the problem.

Li and Womer (2009b) consider a slightly modified version of the problem where the skill sets of workers are predefined and costs have to be minimized that are incurred for each worker used to staff the project. In their computational study, they chose identical costs for all workers and thus minimized the number of required workers. To solve the cost

minimization problem, Li and Womer (2009b) developed a two-phase decomposition approach. The first phase determines a minimum-cost solution to the assignment part of the problem by solving a binary program using CPLEX. The second phase uses a constraint programming approach to determine a feasible schedule for the given assignment. If a feasible schedule does not exist, cuts are derived and added to the first-phase assignment problem and the procedure starts anew. It is repeated until a feasible schedule has been found in the second phase.

The cuts that are derived from an assignment solution which cannot be extended to a feasible schedule include globally and locally valid cuts. Globally valid cuts exclude only infeasible solutions of the assignment problem from the search space, whereas locally valid cuts can exclude feasible and optimal solutions from the search space. A global cut derived by Li and Womer (2009b) inhibits that a worker is assigned to two tasks that must run in parallel. A locally valid cut prevents that a worker is assigned to two tasks that can overlap but need not overlap.

The previously mentioned three-phase approach of Li and Womer (2009a) is an add method. We devised a simple add method for the workforce assignment problem but preliminary computational tests marked it inferior compared to our drop method (see also Subsection 6.2.3.1). It may be more promising to devise a more sophisticated add method seizing ideas from Li and Womer (2009a,b). We will elaborate on this idea in the outlook in Chapter 9.

The works considered up to this point have dealt with staffing unrelated tasks or tasks of a single project; in the remainder of this section, we address staffing problems that feature multiple projects. First, we refer to two approaches to multi-project staffing where the number of different types of assignments is limited by constraints. Both approaches deal with single-period problems in which the workload of concurrent projects must be covered. In Patanakul et al. (2007), the problem of assigning project managers to projects is considered. For each manager, two side conditions limit the number of his assignments. The first condition implies an individual upper bound on the number of concurrent projects that a manager can head effectively. The second condition takes into account that a manager who is assigned to more than one project needs time to switch over from one project to another. The total switch-over time required by a manager increases with the number of projects that he leads. The second constraint considers this switch-over time of a manager together with the workload of his projects and ensures that his availability is not exceeded. Patanakul et al. have formulated the assignment problem as a MIP and use a commercial solver to obtain a solution.

In Certa et al. (2009), skill requirements of projects must be allocated to multi-skilled workers with heterogeneous skill levels. A greater skill level does not reduce the time needed to accomplish a certain amount of workload but increases outcome quality. To maximize overall outcome quality is the first out of three objectives. The second objective is related to learning that occurs whenever a skill is executed. For each skill, the average skill level across all workers that results at the end of the planning period is considered; the minimum of these resulting average skill levels is maximized. The third objective maximizes the preferences for collaboration over all pairs of workers. Here, additional assignments of workers to skill requirements can increase the objective function value. Though, three types of constraints limit the number of assignments for workers to the skill requirements of projects.

The first type of constraints prescribes that the contribution of a worker to a skill requirement must be either zero or greater than a predefined percentage of the workload. The reason for these constraints is the assumption that a small contribution hampers the efficient use of an employee's working time. In order to concentrate a worker's attention on a few projects, the second type of constraints limits the number of projects to which each worker can contribute. The third constraint set demands that each skill requirement must be accomplished by at most two workers in order to avoid scattering workload across workers.

Certa et al. have devised a two-step approach to solve their nonlinear staffing model. In the first step, various methods are applied to generate non-dominated solutions for the multi-objective problem. In the second step, the ELECTRE III method[1] is used to rank the solutions according to the preferences of a decision maker.

We refrain from modifying our original formulation of the workforce assignment problem by adding an upper bound on the size of each project team because these bounds could render an optimal solution of the original problem infeasible. In case of an instance with high workforce utilization, it is quite difficult to specify consistent upper bounds for all projects that are reasonably small but not so restrictive that the resulting set of feasible solutions is empty.

Multi-period multi-project staffing problems are considered by Grunow et al. (2004) and Heimerl and Kolisch (2010a). Grunow et al. (2004) consider a problem where clinical studies must be scheduled over a day such that the tasks of all studies can be satisfied by properly qualified employees. Scheduling decisions must only be taken for the start times of studies. The start times of the tasks that belong to a study are coupled to the beginning of the study and are automatically fixed as soon as the study start time has been chosen. For each task and each period of task duration, a predefined number of qualified employees must be assigned to the task and those employees that contribute to a task must be assigned to it for its complete duration. Each employee is presumed to be qualified for a subset of all tasks. He can be assigned to at most one task in each period and can only work in those periods that belong to the shift to which he is assigned. Each shift spans a set of contiguous periods and an employee can be assigned to at most one shift. If an employee is assigned to a shift, costs are incurred that depend on the employee or his qualification and on the shift. The goal is to find a schedule and an assignment that minimize the sum of fixed costs incurred for required employees.

Grunow et al. apply a hierarchical planning approach and decompose the problem into two subproblems. In the subproblem at the first stage, only the start times of the studies and the duty periods of employees are determined but it is ensured that a feasible assignment of employees to tasks will exist. To ensure this feasibility, Grunow et al. represent task requirements, availabilities of employees, and availabilities of their qualifications in an aggregated fashion. At the second stage, an admissible assignment must be determined. For this feasibility problem, the goal is introduced to minimize the number of assignments of employees to studies, i.e., an employee shall concentrate his contributions on tasks that belong to as few studies as possible. Hence, the average size of a team that accomplishes the tasks of a clinical study is minimized.

In the following, we present the optimization model of the second-stage problem be-

[1] ELECTRE is the abbreviation for *élimination et choix traduisant la realité* which means *elimination and choice expressing reality.*

cause this problem is quite similar to our workforce assignment problem and because we want to check whether it is NP-hard. Grunow et al. (2004, p. 313) stress that their second-stage "assignment model is computationally very efficient due to its pure binary nature". The good computational behavior is underpinned by the results of their numerical study. Though, their problem is obviously related to strongly NP-hard problems such as MINIMUM COVER. Since Grunow et al. do not analyze the complexity of their assignment problem, we are interested in settling the complexity question.

For the presentation of the model, let $\mathcal{P}$ denote the set of studies; $\mathcal{J}$ the set of tasks; $\mathcal{K}$ the set of scheduled employees; $\mathcal{K}_j$, $j \in \mathcal{J}$, the set of those employees that can contribute to task j because they are qualified for task j and they are on duty during the periods in which task j must be performed; and let $\mathcal{J}^{\text{overlap}}$ denote the set that contains all pairs (j, j') of tasks whose processing intervals overlap.

Two types of parameters are given and two types of decision variables are used. The parameter a_j, $j \in \mathcal{J}$, indicates the number of employees required to perform task j and the parameter α_{pj}, $p \in \mathcal{P}$, $j \in \mathcal{J}$, is equal to 1 if task j belongs to study p, and 0 otherwise. The binary decision variable x_{kj}, $j \in \mathcal{J}$, $k \in \mathcal{K}_j$, takes on a value of 1 if employee k is assigned to task j, and a value of 0 otherwise. Likewise, the binary decision variable y_{kp}, $k \in \mathcal{K}$, $p \in \mathcal{P}$, equals 1 if employee k contributes to study p. With these identifiers, the second-stage problem of Grunow et al. (2004, section 4.2) reads as follows.

$$\text{Min.} \quad \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}} y_{kp} \tag{5.1}$$

$$\text{s. t.} \quad \sum_{k \in \mathcal{K}_j} x_{kj} = a_j \qquad j \in \mathcal{J} \tag{5.2}$$

$$x_{kj} + x_{kj'} \leq 1 \qquad (j, j') \in \mathcal{J}^{\text{overlap}},\ k \in \mathcal{K}_j \cap \mathcal{K}_{j'} \tag{5.3}$$

$$y_{kp} \geq x_{kj} \alpha_{pj} \qquad j \in \mathcal{J},\ k \in \mathcal{K}_j,\ p \in \mathcal{P} \tag{5.4}$$

$$x_{kj} \in \{0, 1\} \qquad j \in \mathcal{J},\ k \in \mathcal{K}_j \tag{5.5}$$

$$y_{kp} \in \{0, 1\} \qquad k \in \mathcal{K},\ p \in \mathcal{P} \tag{5.6}$$

Objective function (5.1) minimizes the number of assignments of employees to clinical studies. Constraints (5.2) ensure that the required number of properly qualified employees is assigned to each task. Constraint set (5.3) guarantees that no employee has to perform two or more tasks simultaneously. Constraints (5.4) record the assignments of employees to studies. The domains of the decision variables of this staffing subproblem are defined in Constraint sets (5.5) and (5.6).

Let us term optimization problem (5.1)–(5.6) the *multiple clinical studies staffing problem* ($\text{MCSSP}_{\text{opt}}$) and let us abbreviate the decision problem that corresponds to $\text{MCSSP}_{\text{opt}}$ by $\text{MCSSP}_{\text{dec}}$. Decision problem $\text{MCSSP}_{\text{dec}}$ asks whether there is a solution with at most b assignments of employees to studies, $b \in \mathbb{N} \setminus \{0\}$. In the following, we will show that $\text{MCSSP}_{\text{opt}}$ is NP-hard in the strong sense by reduction from the fixed job scheduling problem with machine availabilities.

Lemma 5.1 $\text{MCSSP}_{\text{dec}} \in \text{NP}$. □

PROOF Let x_{kj}, $j \in \mathcal{J}$, $k \in \mathcal{K}_j$, and y_{kp}, $k \in \mathcal{K}$, $p \in \mathcal{P}$, be a (guessed) solution for an arbitrary instance of $\text{MCSSP}_{\text{dec}}$. We can check in polynomial time if the solution is

feasible and if the number of assignments of workers to studies that is associated with this solution does not exceed b. A check whether the solution is feasible requires to test if the solution satisfies Constraint sets (5.2)–(5.6). The number of constraints within these sets is bounded by a polynomial in the instance size. ■

Theorem 5.1 $\text{MCSSP}_{\text{dec}}$ *is strongly* NP*-complete.* □

Proof (Pseudopolynomial transformation from FJSP with Machine Availabilities)
Problem: FJSP with Machine Availabilities
Instance: Set of n jobs where each job $j = 1, \ldots, n$ must be processed in the interval $[s_j, f_j)$; set of m machines where each machine $i = 1, \ldots, m$ is continuously available in the interval $[a_i, b_i)$.
Question: Does there exist a feasible schedule that uses not more than m machines, i.e., does there exist an assignment of all jobs to the m machines such that each job is processed by one machine and no pair of jobs whose intervals overlap is processed by the same machine?

FJSP with Machine Availabilities is NP-complete in the strong sense (cf. Kolen et al., 2007, section 4.1).

We restrict $\text{MCSSP}_{\text{dec}}$ to the special case where only one study must be staffed, each task requires only one worker, and each worker is qualified to perform every task. That is, we assume $|\mathcal{P}| = 1$; $a_j = 1$, $j \in \mathcal{J}$; and that the set $\mathcal{K}_j$, $j \in \mathcal{J}$, contains all employees whose availability periods cover the processing interval of task j.

Now, let each job j of an instance of the FJSP with Machine Availabilities be associated with a task of the study and let the processing interval of each task be defined by the processing interval $[s_j, f_j)$ of the respective job j. Associate each machine i with an employee and let the availability interval $[a_i, b_i)$ of machine i define the duty period of the respective employee. Then, there exists a feasible assignment of jobs to machines that uses at most m machines if and only if there is a feasible assignment of employees to the tasks of the sole study such that at most m employees are assigned to the study. Together with Lemma 5.1 our proof is complete. ■

Corollary 5.1 $\text{MCSSP}_{\text{opt}}$ *is* NP*-hard in the strong sense.* □

Although their second-stage staffing subproblem $\text{MCSSP}_{\text{opt}}$ is strongly NP-hard, Grunow et al. (2004) can solve most of their 11 test instances for both the first- and second-stage problem with the exact branch-and-cut method of CPLEX 8.1 within minutes. The instances comprise up to 14 studies, 593 tasks, 480 periods of 3 minutes length, and 54 employees. Grunow et al. ascribe the good computational behavior of their staffing subproblem to its pure binary nature.

Despite many similarities, our workforce assignment problem differs from $\text{MCSSP}_{\text{opt}}$ in several points. First, in $\text{MCSSP}_{\text{opt}}$ the number of employees required for a task is predefined, whereas the number of workers necessary for a skill requirement is not prescribed in the workforce assignment problem. Second, the workforce assignment problem is modeled as a mixed-integer program; it cannot be modeled as a pure binary program. Third, in $\text{MCSSP}_{\text{opt}}$ no skill levels are distinguished. Finally, in $\text{MCSSP}_{\text{opt}}$ an employee can contribute only to one task in a period, whereas a worker can contribute to several skill requirements in a period in the workforce assignment problem. Consequently, it seems

very likely that it will be more time-consuming to solve an instance of the workforce assignment problem than to solve a comparable instance of $\text{MCSSP}_{\text{opt}}$. In case of the workforce assignment problem, it seems also very likely that we have to rely on heuristics for instances of realistic size with more than 100 workers, 50 projects, and 10 skills.

Another strongly NP-hard multi-project scheduling and staffing problem is modeled by Heimerl and Kolisch (2010a). The model describes a problem that originates from the IT department of a semiconductor manufacturer. As in the model of Grunow et al. (2004), start periods of projects must be determined, while start times of activities are coupled to their respective project start time; thus, all skill requirements of a project are scheduled as soon as the project start time has been fixed. As in our model, an arbitrary subset of workers can accomplish a skill requirement and skill levels are distinguished. Skill levels of workers lie in the interval $[0.5, 1.5]$. Heimerl and Kolisch (2010a) pursue the goal of minimizing costs for workers. The costs for a worker depend on the amount of time the worker contributes to skill requirements. Heimerl and Kolisch distinguish between regular working time and overtime, which is more expensive. Outsourcing of workload to external resources is also possible, but it is relatively expensive and internal resources must provide a prescribed share of the total contribution to a skill requirement. Test instances with 20 projects, each with 2 potential start time periods on average; 100 workers; 25 skills, 10 skills per worker, 4 skills per project; and 12 periods were solved by the MIP solver of CPLEX 10.1 within a few minutes.

The workforce assignment problem and the problem outlined by Heimerl and Kolisch (2010a) share many commonalities. However, there are two important differences: The workforce assignment problem does not include scheduling decisions and, more important, its objective function minimizes fixed costs that are incurred when a worker contributes to a project—how much so ever—and that are identical for each combination of a project and a worker. In contrast, Heimerl and Kolisch (2010a) minimize variable costs that are linear in the amount of time that a worker contributes to a skill requirement. Their objective function leads to relatively large project teams, as is demonstrated in an exemplary comparison at the beginning of Section 7.3.

In a follow-up paper, Kolisch and Heimerl (2012) consider an almost identical problem where not only project start times but also activity start times are subject to optimization. Kolisch and Heimerl (2012) devise a solution approach that includes a genetic algorithm followed by a tabu search heuristic. The two meta-heuristics are used to generate schedules. In the genetic algorithm, a schedule is coded as a chromosome whose genes take on integer values. Each integer value represents the realized start-to-start time lag between an activity and its direct predecessor. In the tabu search procedure, a schedule is represented by a vector of activity start times. Each schedule, independent from its representation, implies a staffing subproblem, which is a linear program. To solve these staffing subproblems, Kolisch and Heimerl (2012) apply the generalized network simplex method. For their numerical analysis, Kolisch and Heimerl (2012) solved instances with up to 10 projects, 6 activities per project, activity start time windows of 5 periods length, 10 workers, and 4 skills.

The coding schemes of both meta-heuristics outlined by Kolisch and Heimerl (2012) refer to the scheduling part of their problem and thus cannot be simply adapted to our assignment problem. However, we also use the generalized network simplex method to

solve a subproblem that is similar to their staffing subproblem (see Subsections 6.2.3.2 and 6.2.3.3).

To the best of our knowledge, the problem of assigning multi-skilled workers with heterogeneous skill levels to projects such that average team size is minimized has not been considered so far. Though, our literature review demonstrates that related problems have been tackled. The staffing subproblem presented by Grunow et al. (2004) comes closest to the workforce assignment problem. However, Grunow et al. have not outlined heuristic solution methods for their problem. For large-sized instances of the workforce assignment problem, heuristics are needed. Kolisch and Heimerl (2012) have outlined two meta-heuristics for a problem that is also similar to the workforce assignment problem but integrates scheduling decisions and does not take project team size into account. The meta-heuristics, which are adjusted to their problem, cannot be readily applied to our assignment problem. However, an exact method to solve a staffing subproblem was adopted from Kolisch and Heimerl (2012). Beyond that, other solution approaches which we have summarized in this review have—as indicated throughout this section—inspired our solution approaches, which we present in Chapter 6.

5.3 Work related to the utilization leveling problem

In this section, we provide a brief review of contributions that are concerned with leveling the utilization of resources, which is a hard problem in general. We have noted in Section 3.4 that utilization can be measured in absolute and relative terms but that it does not matter for our methods which of both ways is applied. Hence, we do not distinguish whether utilization, workload, or hours worked are leveled and use these terms interchangeably in this section. We have already mentioned several alternative objective functions for leveling problems in Section 4.4. Here, we cover some of those approaches to leveling that deal with projects or multi-skilled workers. Although our utilization leveling problem is a rather easy problem that can be solved by a standard linear programming solver and although the problem does not involve skills because only departmental workload must be allocated, the following review is valuable for two reasons. First, it emphasizes the importance of leveling for firms and their workers. Second, it provides an overview of approaches that may be adopted if the goal is pursued to find an allocation of project and departmental workload that levels the hours worked by employees. The review is divided into two parts. In the first part, we present leveling problems that require scheduling decisions. In these problems, workload can be shifted from one period to another by manipulating activity start times. In the second part, we consider leveling problems where the total workload in each period is already fixed but must be distributed evenly across resources, as it is the case in our problem. In both parts, we begin with works that stress the importance of leveling. Then, we address works that are related to project management or multi-skilled resources.

Lee and Ueng (1999) emphasize that leveling the workload of employees is crucial in order to satisfy the desire for equity and fairness. They consider a vehicle routing problem with a single depot and a fleet of vehicles that must deliver a homogeneous good to customers. The objective function of their optimization model is a weighted sum of two terms, which are both minimized. The first term represents the classical goal of

minimizing the total travel time of all vehicles; the second term targets at leveling the travel times of the vehicles. The travel times of the vehicles are assumed to correspond to the working times of the respective drivers. For each driver, the difference between his travel time and the minimum travel time among all drivers is determined. The sum of these differences constitutes the second term of the objective function. To solve their problem, Lee and Ueng devise a construction heuristic where the next customer that is added to a route is selected according to a savings calculation and assigned to a vehicle whose current route is of minimum length.

The leveling goal of Lee and Ueng, which is represented by the second term in their objective function, is equivalent to the objective of minimizing the "average difference from minimum workload". The balancing effectiveness of this objective has been proven to be rather low (cf. Kumar and Shanker, 2001, Sections 2.1 and 4). To demonstrate the low balancing effectiveness, we consider two solutions of an instance with three drivers whose travel times are $(0, 15, 15)$ and $(0, 10, 20)$, respectively. Although the first solution is intuitively perceived as better leveled, the objective function of Lee and Ueng (1999) signals indifference because the leveling term is equal to 30 for both solutions.

In the realm of project scheduling, resource leveling has a long tradition, but most approaches consider leveling problems that feature mono-skilled resources with homogeneous skill levels. In these problems, the load of a resource must be leveled over time. Often, a quadratic objective function is used because it guarantees effective balancing. For a classical resource leveling problem with a quadratic objective function, Rieck et al. (2012) outline two linearizations that lead to mixed-integer linear programming formulations. For the same problem, Gather (2011) provides a problem-specific branch-and-bound algorithm. In his literature review, several references to other exact methods and to heuristic solution approaches can be found (cf. Gather, 2011, section 1.4).

A project scheduling problem that features not only multi-skilled workers who master skills at different levels but also a leveling objective is presented by Valls et al. (2009). They consider a problem that arises in call centers. When customer inquiries and the related tasks cannot be dealt with by first-level call center agents, these tasks must be assigned to second-level service workers and scheduled subject to general temporal constraints including due dates. Due dates result from service agreements with business clients. Each task must be processed by one service worker who masters the required skill. The service workers differ in their skill sets and can be assigned to at most one task in each period. Three skill levels are distinguished; a higher level reduces task processing times.

For this problem, three goals are pursued in lexicographical order. The most important goal is to schedule the tasks such that priorities are observed as well as possible. The second objective is to assign a highest-skilled worker to each task. The last objective is the leveling objective: For each worker, his total workload over the planning horizon is determined. The leveling objective function minimizes the sum of the absolute deviations of all these workloads from the average workload. To solve the scheduling and staffing problem with the three mentioned objectives, Valls et al. outline a genetic algorithm where scheduling and staffing decisions are encoded as priority lists, which are decoded by a serial schedule generation scheme.

The balancing effectiveness of the leveling objective applied by Valls et al. is only moderate. To illustrate this moderateness, consider four workers and let their workloads

in two solutions be given by $(9, 9, 11, 11)$ and $(8, 10, 10, 12)$, respectively. By intuition, one may rate the first solution as better leveled. However, the leveling objective function of Valls et al. is indifferent towards both solutions.

A scheduling problem where the workload of teams must be leveled is presented by Ho and Leung (2010). They consider the problem of transporting meals for airline passengers from an airport building to aircrafts that are parked at the gates or on the tarmac of the airport. Teams of two workers must deliver the meals regarding the aircrafts' time windows, which result from the flight schedule. Each worker can serve only the aircraft types he is qualified for and an aircraft, which can be interpreted as a customer in a delivery problem, must be served by a team in which at least one worker has the skill to load this aircraft. The problem is to form two-person teams and to assign a route to each team such that a weighted sum of several objectives is minimized. The most important goal is to minimize the number of aircrafts that are not part of a route and are hence not served. Another goal is to level the number of aircrafts that have to be visited by the teams. This goal is incorporated by a quadratic objective function. Ho and Leung solve the catering problem by a two-stage heuristic. At the first stage, a matching approach is used to form teams such that the joint skill sets of the teams are as large as possible and such that rare skills are not accumulated within teams but distributed across teams. At the beginning of the second stage, a route is composed for each team. This initial solution is improved by a tabu search procedure or, alternatively, by a simulated annealing approach.

In situations where a schedule has already been fixed such that the total workload in each period is known and where only decisions about workload allocation must be taken, leveling the workload over resources and leveling the workload of single resources over time, remains a hard problem in general. This hardness can be observed for three out of the four works that we consider in the remainder of this review. Only the problem of Slomp and Molleman (2002) can be modeled as a linear program (LP) and can thus be classified as easy.

Huang et al. (2006) address the problem of assigning airlines to the cargo terminals of an international airport. The cargo flights of an airline are associated with a known amount of workload for loading, unloading, and handling operations. To ease airport operations, all cargo flights of an airline must be assigned to the same cargo terminal. Huang et al. consider a problem with two identical terminals and formulate a MIP model with a leveling objective function that is a weighted sum of two balancing goals. The first goal is to minimize the peak workload that occurs among both terminals over all periods. The second goal is related to the absolute workload difference between both terminals in each period. The sum of these differences shall be minimized. Furthermore, a side condition demands that the workload that is allocated to one of the two terminals over all periods must make up between 45 % and 55 % of the total workload. Huang et al. also formulate a model where workloads for airlines are uncertain. The authors assume that a set of possible scenarios is known and that each scenario specifies a workload for each airline and period. In this case, their goal is to find an assignment of airlines to terminals such that the expected value of the leveling objective function is minimized. To solve large-sized instances of this problem, Huang et al. apply a Benders decomposition approach.

Huang et al. emphasize two advantages of balancing workloads in cargo handling.

First, balancing workloads between terminals avoids congestion at one terminal, while another one is void. Second, balancing workloads at a terminal over time allows the terminal management to efficiently operate with a constant supply of labor, i.e., with constant workforce levels and with regular working times. When loads are not leveled over time, workers may be idle in one period and overburdened in another. Both advantages of leveling hold in principle also for other situations.

A project staffing problem that takes workload leveling into account is presented by LeBlanc et al. (2000). In their problem, managers must be assigned to construction projects. Each project requires one manager, while each manager can lead more than one project. The workload that arises from assigning a manager to a project depends on the travel distance between the manager's home and the site of the construction project. The goal of the corresponding MIP model, which LeBlanc et al. have formulated, is to find an assignment that minimizes the total workload of all managers. The actual leveling is accomplished via constraints of the MIP model. In these constraints, the workload of each manager is restricted to lie between a lower and an upper bound in each period. The bounds are loose in the first optimization run. As long as a feasible solution is found, the bounds are tightened (the lower bounds are increased by one unit, the upper bounds are decreased by one unit) and the model is resolved. LeBlanc et al. select the solution that was found for the tightest bounds as the final solution.

Slomp and Molleman (2002) consider a workload allocation problem with multi-skilled workers in order to assess different cross-training strategies. One criterion for the assessment of a strategy is the workload balance that the strategy facilitates. In their problem, the skill requirements of a single period must be allocated to workers who differ in their skill sets and whose availability is unlimited. Skill levels are not distinguished. Slomp and Molleman outline a three-step approach, which serves different purposes. In the first step, they solve an LP in order to determine an allocation that minimizes the workload of a bottleneck worker. A bottleneck worker is a worker to whom the maximum workload among all workers has been allocated. The second step aims explicitly at leveling the workload across workers. A bottleneck worker of the first-step solution is identified; this worker and his workload are removed from the LP and the reduced LP is solved, again with the objective of minimizing the workload of a bottleneck worker. The removal of a bottleneck worker and his workload and the reoptimization of the reduced LP are repeated until a single worker remains. For the case that a solution features more than one bottleneck worker, Slomp and Molleman do not specify which of them is removed. However, a bad choice can lead to a suboptimal solution. The purpose of the third step is to adapt a solution when a change in skill requirements occurs. The goal is to determine a new solution in which workers execute preferably only those skills that they had to execute in the old solution, because then the need for coordination is assumed to be relatively low.

A problem where multi-skilled employees with heterogeneous skill levels must be assigned to tasks is modeled by Eiselt and Marianov (2008). In their problem, each task needs some skills and for each needed skill a required skill level is specified. Only an employee who masters all needed skills and who meets the required skill levels can be assigned to the task. The skill levels of an employee do not affect the processing time of a task but are related to the boredom that he experiences when executing a task. The greater the distance of his skill levels from the required levels, the less challenging is the task and the greater is the boredom of the employee. The assignment decisions are

evaluated by an objective function that is a weighted sum of three goals. One goal is to minimize the boredom of employees. A second goal is to minimize costs for overtime and outsourcing of tasks. These costs may be incurred because all tasks must be accomplished but the regular working time and extra hours of each employee are limited. The third goal is to balance the workload of employees. For each employee, Eiselt and Marianov determine the absolute difference between his workload and the average workload. The sum of all differences is minimized. The whole problem is formulated as a MIP and solved with CPLEX.

This review of approaches to workload leveling underlined the importance of workload balancing in practice and justifies our aim to level worker utilization by an appropriate allocation of departmental workload. Furthermore, the review revealed that workload leveling is often not the goal with highest priority. Frequently, it is only one goal among others or a secondary goal. This holds also for our approach, where leveling is placed at the last stage of the hierarchical planning model.

Chapter 6

Solution methods

In this chapter, we will outline solution methods for the three optimization problems modeled in Chapter 4. We concentrate on solution methods for the problem of assigning workers to projects, because this problem is in the focus of our work. Methods for the selection and the leveling problem will be considered only in brief.

We present exact and heuristic solution methods. Exact methods determine an optimal solution to a problem instance if an optimal solution exists; otherwise, they prove that no optimal solution exists. Heuristic solution methods apply promising rules to find solutions, but do not explore the whole solution space. Heuristics are supposed to find good solutions in short time. Heuristic methods cannot guarantee to find an optimal solution if an optimal solution exists, and may even fail to find an existing feasible solution for an instance. The major advantage of heuristic solution methods, however, is their speed when large-sized problem instances are tackled; for such instances, heuristics tend to be much faster than exact methods. Furthermore, in case of hard problems, good heuristic methods tend to provide better solutions than exact methods when solution time is limited. Solution time is often limited, because problem owners, e.g., a firm's top management as in our case, tend to prefer short solution times.

In regard to our three optimization problems, heuristic solution methods come into question for the selection and the assignment problem, as these are hard problems. The complexity analysis in Section 4.6 revealed that the project selection problem and the problem of assigning workers to projects are NP-hard in the strong sense. This complexity result implies that the time that is necessary to determine an optimal solution for these problems is exponential in the instance size. This exponential time complexity may render exact solution methods for these two problems unattractive when it comes to large-scale instances. Furthermore, strong NP-hardness of a problem implies that no *fully polynomial-time approximation scheme* exists for this problem (cf. Garey and Johnson, 1979, pp. 121–151, especially pp. 140–142; Wegener, 2003, pp. 105–116, especially pp. 114–115; Gomes and Williams, 2005). This means that for our strongly NP-hard problems no heuristics exist that can guarantee for any instance to find a solution with an arbitrarily small relative gap ϵ and that have a running time bounded by a polynomial in the instance size and in $\frac{1}{\epsilon}$.

Nevertheless, we will outline exact methods not only for our leveling problem but also for our selection and our assignment problem for two reasons. First, for instances of small size, the solution time of exact methods may be acceptable for the problem owners. Even for large-sized instances, the solution quality that is reached within a certain time limit can be acceptable. Second, exact methods are required for an assessment of heuristic

M. Walter, *Multi-Project Management with a Multi-Skilled Workforce,* Produktion und Logistik, DOI 10.1007/978-3-658-08036-5_6, © Springer Fachmedien Wiesbaden 2015

methods, because exact methods can provide optimal solutions for small-sized instances. These solutions can serve as a benchmark for solutions found by heuristic methods.

In Section 6.1 we present exact solution methods for our three problems. For the project selection problem and for the workforce assignment problem, we resorted to a solver package that offers a branch-and-cut method. In preliminary tests, we found that this exact method provided satisfactory results for the project selection problem. For the workforce assignment problem, we devised valid inequalities to support the branch-and-cut method. Finally, we present a specially tailored exact algorithm for the utilization leveling problem.

Section 6.2 is devoted to heuristic methods for the workforce assignment problem. Since the exact methods did not provide satisfying results for this problem, we developed four heuristic methods, which differ in the extent to which solvers for mixed-integer linear programs and linear programs are used.

6.1 Exact solution methods and their support

In this section, we will state the exact solution methods that we applied to our three optimization problems and describe measures that support these methods. Our three optimization problems comprise the project selection problem and the workforce assignment problem, which were modeled as mixed-integer linear programming (MIP) models, and the utilization leveling problem, which was modeled as a linear program (LP).

An exact state-of-the-art approach to solve LPs is the simplex method. An exact state-of-the-art approach to solve MIP models are branch-and-cut methods. Branch-and-cut methods combine classical branch-and-bound and cutting plane algorithms.

For the following, we presume that the reader is familiar with the simplex method. For a comprehensive treatment of the simplex method, see Dantzig (1963), Chvátal (1983), or Neumann and Morlock (2002, pp. 35–171), for example. The simplex method is also applied as a subroutine when MIP models are solved by branch-and-bound or branch-and-cut.

We also presume that the reader is familiar with the principle of branch-and-cut methods. The two main building blocks of branch-and-cut methods are branch-and-bound and cutting plane algorithms. For a thorough treatment of the branch-and-bound method we refer to Nemhauser and Wolsey (1999, pp. 349–367), Winston and Goldberg (2004, pp. 475–545), and Neumann and Morlock (2002, pp. 392–402). Cutting plane algorithms try to generate cutting planes, which are specific valid inequalities. Both valid inequalities and cutting planes are used to tighten the LP relaxation. At the beginning of Subsection 6.1.1, we will briefly review valid inequalities and cutting planes. Integrating cutting plane algorithms into branch-and-bound methods has, in general, a positive impact on solution times, as documented in Bixby and Rothberg (2007). The integration of cutting plane algorithms into branch-and-bound methods leads to branch-and-cut methods and is described by Wolsey (1998, pp. 157–160) and Neumann and Morlock (2002, pp. 529–535). Examples for branch-and-cut methods can be found in Obreque et al. (2010) and Gicquel et al. (2010).

Both the simplex method and branch-and-cut methods are well-established and have been included in readily available solver packages. For example, the commercial solver

package CPLEX from IBM ILOG features LP solvers which are based on the simplex method and a MIP solver which is based on a branch-and-cut method. With regard to the LP solvers, CPLEX offers the primal and the dual simplex method among others. A preliminary comparison between the LP solvers of CPLEX for our solution methods revealed that there is no significant difference in solution speed. Hence, we chose to apply the default LP solver of CPLEX, which is the dual simplex method, for our computational tests. Whenever we refer to *the LP solver of CPLEX* in the following, we mean the dual simplex method of CPLEX; the term *the simplex method* includes both the primal and the dual simplex method.

We used the branch-and-cut method offered by CPLEX to solve the project selection problem. As the results obtained with CPLEX were satisfactory, we did not see the need for additional effort and we abstained from measures supporting CPLEX and also refrained from developing heuristic methods.

We also applied the branch-and-cut method offered by CPLEX to the MIP model of the workforce assignment problem. The results obtained were only satisfactory for instances of small size. In order to support the branch-and-cut procedure, we devised valid inequalities that tighten up the formulation of the MIP model for the workforce assignment problem. In Subsection 6.1.1, we will succinctly introduce the concept of valid inequalities and their potential for cutting planes, and then present our valid inequalities for the workforce assignment problem and discuss their potential acting as cutting planes.

In Subsection 6.1.2, we will present an exact solution method for the utilization leveling problem, i.e., for the problem of leveling working times by allocating departmental workload. For this LP, the simplex method can serve as an exact method. However, the worst-case running time of the simplex method is not polynomial. This time complexity result holds for all variants of the simplex method such as the primal and dual variant. Alternative general-purpose methods for LPs exist that have polynomial time complexity. Though, in practice the simplex method performs well and often outpaces the polynomial-time methods. We will present a polynomial-time algorithm for our leveling problem whose worst-case running time is shorter than the worst-case running time of the general-purpose polynomial-time algorithms for LPs. Furthermore, our algorithm outperforms the simplex method when applied to test instances that come close to real-life instances.

6.1.1 Valid inequalities for the workforce assignment problem

This subsection starts with a succinct review of *valid inequalities* and *cutting planes*. The idea that lies behind these two types of inequalities is to enhance the solution of mixed-integer linear programming (MIP) models and pure integer linear programming (PIP) models by tightening the model formulation, i.e., by tightening the feasible region of the LP relaxation. At the end of the review, we will distinguish local cuts and locally valid inequalities from global cuts and globally valid inequalities, respectively.

Then, we will turn to the workforce assignment problem, for which we formulated MIP models, and derive three types of valid inequalities that tighten its LP relaxation. The first type of valid inequalities addresses the minimum number of workers that is necessary to satisfy the requirements of a project p for a skill $s \in \mathcal{S}_p$. The second type addresses the minimum number of workers that is necessary to satisfy all requirements of a project p.

Finally, the third type of valid inequalities are tighter formulations of big-M constraints of the standard and the network model. For all three types of valid inequalities, we will present globally valid inequalities. For the first two types of valid inequalities, we will also present locally valid inequalities. At the end of this subsection, we will briefly discuss the potential of the presented inequalities for acting as cutting planes.

For our review of valid inequalities and cutting planes, we consider problems that can be formulated either by a MIP or a PIP model. Let Π be an arbitrary problem modeled as a MIP or a PIP. May S denote the set of feasible solutions for Π and may $P \supseteq S$ denote the feasible region for the LP relaxation of problem Π. P is also termed the search space of the LP relaxation.

A constraint that is satisfied by all points in S is called a *valid inequality* for S (cf. Nemhauser and Wolsey, 1999, p. 88). Adding valid inequalities to the constraint set of a problem can significantly reduce the number of LP relaxations that must be solved and the number of branching steps that are necessary in order to find an optimal solution for the problem.

To give an example for a valid inequality and to indicate how a valid inequality can reduce the number of LP relaxations that must be solved, we consider the knapsack problem, which was introduced on page 76 in Section 4.6.

Example 6.1 Let the set $J = \{1, \ldots, n\}$ contain n items, let the parameter w_j denote the weight of item $j \in J$, and let a knapsack with capacity c be given. A subset of items whose total weight must not exceed the capactiy c has to be selected. The set of feasible solutions is given by $S = \{\boldsymbol{x} \in \{0,1\}^n \mid \sum_{j \in J} w_j x_j \leq c\}$, where the binary decision variable x_j, $j \in J$, indicates whether we select item j ($x_j = 1$) or not ($x_j = 0$). Let $P = \{\boldsymbol{x} \in [0,1]^n \mid \sum_{j \in J} w_j x_j \leq c\}$ denote the feasible region of the corresponding LP relaxation. P is also called knapsack polytope.

To state a valid inequality, let there be a set $C \subset J$ with $\sum_{j \in C} w_j > c$. Such a set C is called a cover. Then, $\sum_{j \in C} x_j \leq |C| - 1$ is a valid inequality for the knapsack polytope (cf. Nemhauser and Wolsey, 1999, pp. 215–216, 265–270; Balas, 1975). This inequality, called cover inequality, states that not all items from C can be selected. Note that a solution $\boldsymbol{x}^{\mathrm{LP}}$ that is feasible for the LP relaxation can imply $\sum_{j \in C} x_j^{\mathrm{LP}} > |C| - 1$. Adding our valid inequality to the knapsack polytope P would exclude the solution $\boldsymbol{x}^{\mathrm{LP}}$ from the resulting tightened polytope and would thus make $\boldsymbol{x}^{\mathrm{LP}}$ infeasible for the tightened LP relaxation. □

The example shows how valid inequalities can tighten a model formulation and how a tight formulation improves the chance to find an integer-feasible solution by excluding integer-infeasible solutions from the feasible region of the LP relaxation.

Transferring this insight to our problem Π leads to the *convex hull* of the solution space S. If problem Π is formulated as tight as possible, the feasible region of the LP relaxation of Π coincides with the convex hull of S. The convex hull of S, denoted by $\mathrm{conv}(S)$, are all points that are convex combinations of points in S (cf. Nemhauser and Wolsey, 1999, p. 83). If the feasible region of the LP relaxation coincides with $\mathrm{conv}(S)$, an optimal solution for the LP relaxation that is determined by the simplex method is also an optimal solution for the original problem Π. Hence, a description of the convex hull $\mathrm{conv}(S)$ is precious, as such a description accelerates the solution process.

For the description of the convex hull of S, special valid inequalities are necessary. Unfortunately, the LP relaxation of a MIP or PIP model is not as tight as possible in general, i.e., $\text{conv}(S) \subset P$ holds in general. Then, a description of the convex hull $\text{conv}(S)$ can be obtained by adding distinguished valid inequalities that are called *facets* to the LP relaxation of problem Π. Let $\text{conv}(S)$ be n-dimensional, then a facet is a $(n-1)$-dimensional face of $\text{conv}(S)$, i.e., a hyperplane that is an $(n-1)$-dimensional boundary of $\text{conv}(S)$ (cf. Nemhauser and Wolsey, 1999, pp. 88–92; Conforti et al., 2010, Chapter 11, Section 2.7).

Although determining all facets for the feasible region of a problem may be too expensive, it can be helpful to determine some facets or weaker valid inequalities and add them to the constraint set. For mixed-integer linear and pure integer linear optimization problems that are NP-hard, no efficient way is known to determine all facets of $\text{conv}(S)$. If an efficient way was known, P = NP would hold. The cover inequalities that we introduced in Example 6.1 on the preceding page are no facets in general (cf. Wolsey, 1998, pp. 147–150; Balas, 1975). Nevertheless these valid inequalities help to accelerate the solution process. Generally, it can be worthwhile to search for valid inequalities that cut off parts from the set $P \setminus S$ in order to tighten the description of S. Such valid inequalities lead to a tighter LP relaxation, which comes closer to the convex hull of S. This idea of tightening the LP relaxation is pursued by cutting planes, which are valid inequalities that are generated on demand.

Cutting planes, also called cuts, are valid inequalities that cut off solutions in $P \setminus S$ from the solution space S of a problem Π. Suppose that a solution to the LP relaxation of problem Π contains fractional values for at least one integer variable. This solution is a point that lies in $P \setminus S$. A cutting plane separates this point from the set S of integer-feasible solutions. If we add such a cutting plane to P, the resulting search space P' is a tighter description of S than P. In this way, cutting planes decrease the size of the search space P and thus accelerate the search for an optimal solution.

This concept of tightening the description of S leads to cutting plane approaches that solve linear problems featuring integer variables by repeatedly solving the LP relaxation and adding cutting planes until the solution of the LP relaxation is integer-feasible. Gomory (1958) introduced such an approach for pure integer programs, where so called Gomory fractional cuts were generated. Again Gomory (1960) presented such an approach for mixed-integer linear programs, where so called Gomory mixed integer cuts were generated (cf. also Balinski, 1965).

Cutting planes are generated by cutting plane algorithms, which try to find a valid inequality that separates a fractional solution of the LP relaxation from S. This problem of determining a violated inequality is called *separation problem*. Each cutting plane algorithm is a procedure that tries to generate cutting planes of a certain class. For example, Gomory (1958) presented a procedure that generates cuts from the class of Gomory fractional cuts. Another class of cuts are cover cuts. Cover cuts are cover inequalities that cut off non-integer points from a knapsack polytope. Nemhauser and Wolsey (1999, pp. 459–460) outline a cutting plane algorithm that generates cover cuts. While cover cuts apply to sets of binary variables only, the class of flow cover cuts applies to mixed-binary sets. Flow cover cuts can be derived from flow conservation constraints at nodes in flow networks (cf. Padberg et al., 1985; Nemhauser and Wolsey, 1999, pp. 281–290).

There exist many other classes of cuts for which cutting plane algorithms have been devised. Many of these cutting plane algorithms have been integrated into branch-and-bound methods to form branch-and-cut methods. These branch-and-cut methods are included in solver packages such as CPLEX.

For branch-and-cut methods as well as for branch-and-bound methods, we must distinguish global from local cuts (cf. Balas et al., 1996) and globally from locally valid inequalities. A cut (valid inequality) that is valid for the root node of a branch-and-cut or branch-and-bound tree is called a global cut (globally valid inequality). A global cut (globally valid inequality) is valid for the original problem Π and all its subproblems, which are created by branching. A local cut (locally valid inequality), in contrast, is only valid for a certain subproblem Π^{sub} and for the subproblems of Π^{sub}. A local cut (locally valid inequality) exploits local information of a node in a branch-and-cut or branch-and-bound tree and is hence only valid for this node and its subnodes.

In the following, we present three types of valid inequalities for the workforce assignment problem. Among these valid inequalities are globally and locally valid inequalities. We derived the three types of valid inequalities, which tighten the LP relaxation of the workforce assignment problem, by exploiting the structure of this problem.

The first type of valid inequalities that we consider is based on a lower bound for the number of workers that are necessary to satisfy the requirements of a project p for a skill $s \in \mathcal{S}_p$ in all periods during project execution. The following example demonstrates our idea.

Example 6.2 Assume that in an instance of the workforce assignment problem only two workers k_1 and k_2 master skill s, which is the only skill required by project p. For $k \in \{k_1, k_2\}$, let $l_{ks} = 1$ and $R_{kt} = 10$. Let project p last only for one period t and let $r_{pst} = 11$. Then, in an optimal solution to the LP relaxation, $x_{k_1p} + x_{k_2p} = 1.1$ holds, e.g., with $x_{k_1p} = 1$, $y_{k_1pst} = 10$, $x_{k_2p} = 0.1$ and $y_{k_2pst} = 1$. Though, it is obvious that $x_{k_1p} + x_{k_2p} \geq 2$ must hold in an integer-feasible solution, because both workers are needed to accomplish the requirements of project p. Here, 2 is a lower bound on the number of workers that must be assigned to project p for skill s. □

A globally valid inequality like the one in Example 6.2 can be derived for each project and each required skill. Let LB_{ps}^{glob}, $p \in \mathcal{P}$, $s \in \mathcal{S}_p$, denote a lower bound for the number of workers that is necessary to satisfy the requirements of project p for skill s. Then, the following set of globally valid inequalities can be formulated.

$$\sum_{k \in \mathcal{K}_s} x_{kp} \geq LB_{ps}^{\text{glob}} \qquad p \in \mathcal{P},\ s \in \mathcal{S}_p \tag{6.1}$$

For each project $p \in \mathcal{P}$ and each skill $s \in \mathcal{S}_p$, Constraint set (6.1) requires that at least LB_{ps}^{glob} workers from those workers who master skill s must be assigned to project p.

The lower bounds LB_{ps}^{glob} in the valid inequalities (6.1) can be calculated by Algorithm 6.1. To compute the lower bound LB_{ps}^{glob} for project p and skill s, we calculate for each period $t \in \mathcal{T}_p$ the minimum number of workers n^* that is necessary to accomplish the requirement r_{pst} (cf. line 12 of Algorithm 6.1). To calculate n^* for a period $t \in \mathcal{T}_p$, we first sort all workers who master skill s in order of non-increasing maximum hours that they can accomplish from the required hours r_{pst}. Then, we assign worker by worker in

this order to project p until the assigned workers can satisfy the complete skill requirement r_{pst}. From the results for n^* for all periods $t \in \mathcal{T}_p$, we select the maximum value as lower bound LB_{ps}^{glob}. Algorithm 6.1 runs in $O(TK^2)$ time if the sorting algorithm *quicksort* is used to sort the vector *vecSkillCap*. Quicksort has a worst-time complexity of $O(K^2)$.

Algorithm 6.1 Calculation of the lower bound LB_{ps}^{glob} on the number of assignments of workers to project p for skill s

```
1: Input: Project p, skill s ∈ S_p, instance data
2: Output: Lower bound LB_ps^glob
3: ---------------------------------------------------------------
4: LB_ps^glob := 0;
5: Declare a vector vecSkillCap of length |K_s|;
6: for all t ∈ T_p do
7:     i := 1;
8:     for all k ∈ K_s do
9:         vecSkillCap[i] := R_kt l_ks;
10:        i := i + 1;
11:    Sort the entries in vecSkillCap in order of non-increasing values;
12:    n* := min {n ∈ ℕ | r_pst − Σ_{i=1}^{n} vecSkillCap[i] ≤ 0};
13:    LB_ps^glob := max(LB_ps^glob, n*);
```

The same idea that led to globally valid inequalities (6.1) also leads to related locally valid inequalities. Let us consider a project p and a skill $s \in \mathcal{S}_p$. Suppose that some variables x_{kp}, $k \in \mathcal{K}_s$, have already been fixed to either 0 or 1, e.g., in the course of a branch-and-cut procedure. Let $\mathcal{K}_s^{p0} \subseteq \mathcal{K}_s$ and $\mathcal{K}_s^{p1} \subseteq \mathcal{K}_s$ denote the set of workers that master skill s and whose variables x_{kp} have already been fixed to 0 and 1, respectively. Let the set $\mathcal{K}_s^{p\text{NF}} := \mathcal{K}_s \setminus (\mathcal{K}_s^{p0} \cup \mathcal{K}_s^{p1})$ contain those workers whose variable x_{kp} has not been fixed so far. The case that the sets $\mathcal{K}_s^{p\text{NF}}$ and $\mathcal{K}_s^{p0} \cup \mathcal{K}_s^{p1}$ are non-empty occurs regularly in nodes of a branch-and-cut tree, for example. In such a case, we can calculate a locally valid lower bound LB_{ps}^{loc} on the number of workers from the set $\mathcal{K}_s^{p\text{NF}}$ that must be assigned to project p in order to accomplish the requirements of project p for skill s. This leads to the following set of locally valid inequalities.

$$\sum_{k \in \mathcal{K}_s^{p\text{NF}}} x_{kp} \geq LB_{ps}^{\text{loc}} \qquad p \in \mathcal{P},\ s \in \mathcal{S}_p \tag{6.2}$$

For each project $p \in \mathcal{P}$ and each skill $s \in \mathcal{S}_p$, Constraint set (6.2) requires that at least LB_{ps}^{loc} workers who master skill s and whose variable x_{kp} has not been fixed so far must be assigned to project p.

The lower bounds LB_{ps}^{loc} in the valid inequalities (6.2) can be calculated by Algorithm 6.2. In Algorithm 6.2, we assume for each period t of project execution that as much workload of project p and skill s as possible is allocated to those workers that have already been assigned to project p (cf. lines 8–9 of Algorithm 6.2). For the resulting remaining requirement r_{pst}^{rem} of project p for skill s in period t, we determine the minimum number of workers n^* that is necessary to satisfy this remaining requirement. We take into account that only workers in $\mathcal{K}_s^{p\text{NF}}$ can satisfy this requirement. From the results for n^*

for all periods $t \in \mathcal{T}_p$, we select the maximum value as lower bound LB_{ps}^{loc}. Algorithm 6.2 requires $O(TK^2)$ time if quicksort is used to sort the vector *vecSkillCap*.

Algorithm 6.2 Calculation of the lower bound LB_{ps}^{loc} on the number of assignments of workers to project p for skill s

Input: Project p, skill $s \in \mathcal{S}_p$, $\mathcal{K}_s^{p0}$, $\mathcal{K}_s^{p1}$, $\mathcal{K}_s^{p\text{NF}}$, instance data
Output: Lower bound LB_{ps}^{loc}

$LB_{ps}^{\text{loc}} := 0$;
Declare a vector *vecSkillCap* of length $|\mathcal{K}_s^{p\text{NF}}|$;
for all $t \in \mathcal{T}_p$ **do**
    $r_{pst}^{\text{rem}} := r_{pst}$;
    for all $k \in \mathcal{K}_s^{p1}$ **do**
        $r_{pst}^{\text{rem}} := r_{pst}^{\text{rem}} - R_{kt} l_{ks}$;
    if $r_{pst}^{\text{rem}} > 0$ **then**
        $i := 1$;
        for all $k \in \mathcal{K}_s^{p\text{NF}}$ **do**
            $vecSkillCap[i] := R_{kt} l_{ks}$;
            $i := i + 1$;
        Sort the entries in *vecSkillCap* in order of non-increasing values;
        $n^* := \min\left\{n \in \mathbb{N} \mid r_{pst}^{\text{rem}} - \sum_{i=1}^{n} vecSkillCap[i] \leq 0\right\}$;
        $LB_{ps}^{\text{loc}} := \max(LB_{ps}^{\text{loc}}, n^*)$;

See that a lower bound LB_{ps}^{loc} calculated by Algorithm 6.2 is equal to the corresponding lower bound LB_{ps}^{glob} if $\mathcal{K}_s^{p0} = \mathcal{K}_s^{p1} = \emptyset$. Furthermore, see that locally valid inequalities (6.2) are at least as tight as globally valid inequalities (6.1), because $LB_{ps}^{\text{loc}} + |\mathcal{K}_s^{p1}| \geq LB_{ps}^{\text{glob}}$ holds.

The second type of valid inequalities that we consider is based on a lower bound for the number of workers that are necessary to satisfy all requirements of a project p. The following example demonstrates our idea.

Example 6.3 Consider an instance of the workforce assignment problem where a project p lasts only for one period t and requires two skills s_1 and s_2. For $s \in \{s_1, s_2\}$, let $r_{pst} = 11$ and $\mathcal{K}_s = \{k_1, k_2, k_3\}$. For each worker $k \in \mathcal{K}_s$, let $R_{kt} = 10$ and $l_{ks} = 1$, $s \in \{s_1, s_2\}$. Assume that globally valid inequalities (6.1) must hold for p and $s \in \mathcal{S}_p$, i.e., $\sum_{k \in \mathcal{K}_s} x_{kp} \geq 2$ must hold for $s \in \{s_1, s_2\}$. Then, in an optimal solution to the LP relaxation, $x_{k_1p} + x_{k_2p} + x_{k_3p} = 2.2$ holds, e.g., with $x_{k_1p} = 1$, $x_{k_2p} = 1$ and $x_{k_3p} = 0.2$. Though, it is obvious that $x_{k_1p} + x_{k_2p} + x_{k_3p} \geq 3$ must hold in an integer-feasible solution, because all three workers are needed to accomplish all requirements of project p. Here, 3 is a lower bound on the number of workers that must be assigned to project p. □

Following the idea of Example 6.3, we can derive globally and locally valid inequalities that are similar to the valid inequalities (6.1) and (6.2), respectively. For the sake of brevity, we will only outline the globally valid inequalities in detail. Let LB_p^{glob}, $p \in \mathcal{P}$, denote a lower bound on the number of workers that is necessary to satisfy all requirements

of project p. Then, the following set of globally valid inequalities can be formulated.

$$\sum_{k \in \mathcal{K}_p^{\text{suit}}} x_{kp} \geq LB_p^{\text{glob}} \qquad p \in \mathcal{P} \tag{6.3}$$

For each project $p \in \mathcal{P}$, inequalities (6.3) require that at least LB_p^{glob} workers from those workers who are suitable for project p must be assigned to project p.

The lower bounds LB_p^{glob} in valid inequalities (6.3) can be calculated by Algorithm 6.3. To compute a lower bound LB_p^{glob}, Algorithm 6.3 determines for each period $t \in \mathcal{T}_p$ the minimum number of workers n^* that is necessary to accomplish all requirements of project p in period t, i.e., to accomplish the sum of all skill requirements of project p in period t. This sum of skill requirements is denoted by r_{pt}. From the results for n^*, the maximum value over all periods is selected as lower bound LB_{ps}^{glob}.

To calculate n^* for a period t, we first calculate for each worker k who is suitable for project p how many hours of the required total hours r_{pt} he can accomplish if he spends all his time for project p. For this calculation, we assume that worker k contributes to matching skills in the order of non-increasing skill levels until his remaining availability R_{kt}^{rem} in period t is depleted. The result for the hours of r_{pt} that worker k can accomplish is stored in the vector *vecProjCap* for each worker $k \in \mathcal{K}_p^{\text{suit}}$. Then, the vector *vecProjCap* is sorted in order of non-increasing values and we assign worker by worker in this order to project p until the assigned workers can satisfy the sum of skill requirements r_{pt}. The number of workers that satisfy r_{pt} is stored in the variable n^*. Algorithm 6.3 runs in $O\left(KS^2 + T(KS + K^2)\right)$ time if quicksort is used for sorting operations.

The locally valid inequalities that correspond to the globally valid inequalities (6.3) read as:

$$\sum_{k \in \mathcal{K}^{p\text{NF}}} x_{kp} \geq LB_p^{\text{loc}} \qquad p \in \mathcal{P} \tag{6.4}$$

The set $\mathcal{K}^{p\text{NF}}$ contains those workers $k \in \mathcal{K}_p^{\text{suit}}$ whose variable x_{kp} has not been fixed so far. From the set $\mathcal{K}^{p\text{NF}}$ at least LB_p^{loc} workers must be assigned to project p in a subproblem. The lower bound LB_p^{loc} can be calculated analogously to Algorithm 6.2.

Both globally valid inequalities, (6.1) and (6.3), are complementary. None dominates the other. This holds also for the corresponding locally valid inequalities. In Example 6.3, it was shown that valid inequalities (6.3) can exclude integer-infeasible solutions from the feasible region of the LP relaxation that are not excluded by valid inequalities (6.1). Vice versa, valid inequalities (6.1) can also exclude integer-infeasible solutions that are not excluded by valid inequalities (6.3), as the following example demonstrates.

Example 6.4 Consider an instance of the workforce assignment problem where a project p lasts only for one period t and requires two skills s_1 and s_2. Let $r_{ps_1t} = 11$ and $r_{ps_2t} = 1$. Assume that only workers k_1 and k_2 master skill s_1 and that only worker k_3 masters skill s_2. Let $l_{k_1s_1} = l_{k_2s_1} = l_{k_3s_2} = 1$ and let $R_{kt} = 10$, $k \in \{k_1, k_2, k_3\}$. Assume that globally valid inequalities (6.3) must hold for project p, i.e., $\sum_{k \in \mathcal{K}_p^{\text{suit}}} \geq 2$ must hold.

Then, in an optimal solution to the LP relaxation, $x_{k_1p} + x_{k_2p} + x_{k_3p} = 2$ holds, e.g., with $x_{k_1p} = 1$, $x_{k_2p} = 0.1$ and $x_{k_2p} = 0.9$. Though, it is obvious that $x_{k_1p} + x_{k_2p} + x_{k_3p} \geq 3$ must hold in an integer-feasible solution, because all three workers are needed to accomplish all requirements of project p. The latter inequality results from $x_{k_1p} + x_{k_2p} \geq 2$ and $x_{k_3p} \geq 1$, which are the valid inequalities (6.1) for project p. □

Algorithm 6.3 Calculation of the lower bound LB_p^{glob} on the number of assignments of workers to project p

Input: Project p, instance data
Output: Lower bound LB_p^{glob}

$LB_p^{\text{glob}} := 0$;
Declare a vector *vecProjCap* of length $|\mathcal{K}_p^{\text{suit}}|$;
for all $t \in \mathcal{T}_p$ **do**
    $r_{pt} := \sum_{s \in S_p} r_{pst}$;
    $i := 1$;
    for all $k \in \mathcal{K}_p^{\text{suit}}$ **do**
        $R_{kt}^{\text{rem}} := R_{kt}$;
        $vecProjCap[i] := 0$;
        for all $s \in \mathcal{S}_{kp}^{\text{match}}$ in order of non-increasing skill levels l_{ks} **do**
            if $R_{kt}^{\text{rem}} > 0$ **then**
                $maxHours := \min(r_{pst}, R_{kt}^{\text{rem}} l_{ks})$;
                $vecProjCap[i] := vecProjCap[i] + maxHours$;
                $R_{kt}^{\text{rem}} := R_{kt}^{\text{rem}} - \frac{maxHours}{l_{ks}}$;
        $i := i + 1$;
    Sort the entries in *vecProjCap* in order of non-increasing values;
    $n^* = \min\{n \in \mathbb{N} \mid r_{pt} - \sum_{i=1}^{n} vecProjCap[i] \leq 0\}$;
    $LB_p^{\text{glob}} := \max(LB_p^{\text{glob}}, n^*)$;

Valid inequalities (6.1) and (6.3) are two members of a greater family of valid inequalities. This family comprises all valid inequalities that can be derived when all combinations of skills in $\mathcal{S}_p$ are considered, i.e., when all $2^{|\mathcal{S}_p|} - 1$ non-empty subsets of $\mathcal{S}_p$ are considered. For each subset of skills, a minimum number of workers can be determined that is necessary to accomplish the requirements of those skills belonging to the subset. To determine the minimum number for a subset of skills, all workers are considered that master at least one skill from the skills in this subset, because these workers can help to accomplish the requirements for the skills in the subset.

The valid inequalities that we presented have a weak point that cannot be remedied easily. The weak point is that our lower bounds LB_{ps}^{glob}, LB_{ps}^{loc}, LB_p^{glob}, and LB_p^{loc} are not very tight. Consider the lower bound LB_{ps}^{glob}, for example. It is determined by Algorithm 6.1 and represents a lower bound on the number of assignments of workers to project p that are necessary to satisfy the requirements of project p for skill s. For our calculation of LB_{ps}^{glob}, we assumed that every worker who masters skill s can spend all his available time to accomplish the requirements of project p for skill s. However, all feasible solutions to an instance of the workforce assignment problem may require that a worker has to spend some time for other skills of project p or for other projects. Our calculation in Algorithm 6.1 ignores this circumstance. If we took such circumstances into account, we would obtain tighter bounds. Though, in our opinion, the effort to incorporate all those circumstances into the calculation of a lower bound is too high. Hence, we have to content ourselves with lower bounds and valid inequalities that might not be very tight.

The same and another, more serious drawback hold for a second family of valid inequalities. We will introduce and discuss this family only in brief. For this family of valid inequalities, we consider the sum of skill requirements over several projects $p \in \mathcal{P}$. Let us introduce two representatives of this family. Both representatives are valid inequalities that take all projects into account. First, a valid inequality can be formulated for each skill $s \in \mathcal{S}$ by computing a lower bound on the number of workers that is necessary to accomplish all requirements of skill s. Second, a valid inequality can be imposed that demands a minimum number of workers required to accomplish the total workload of all projects. Preliminary tests suggested that the family of valid inequalities is not very helpful for boosting the performance of a MIP solver based on branch-and-cut. Let us explain this outcome by an example that refers to both representatives of the family. Consider an instance with $K = P = 10$ and $S = T = 1$ where the sole skill is denoted by s and the unique period by t to simplify notation. Assume that each worker k masters the unique skill s with a level of $l_{ks} = 1$. Let $R_{kt} = 10$, $k \in \mathcal{K}$, and $r_{pst} = 1$, $p \in \mathcal{P}$. Then, one worker is enough to accomplish the total workload of all projects, i.e., a lower bound on the number of assignments equal to 1 is calculated. Though, it is obvious that one worker must be assigned to each project, i.e., that 10 assignments are necessary. In general, this family of valid inequalities tends to be very weak. Hence, we did not apply it.

The third type of valid inequalities that we consider are tighter versions of those big-M constraints that we used in the standard and in the network model in Subsection 4.3.1. All big-M constraints, be it tighter or less tighter versions, are globally valid inequalities. For the sake of a short presentation, we will outline tighter formulations only for the standard model. These formulations can be easily transferred to the big-M constraints of the network model. For the standard model, we presented three alternative big-M constraints: (4.11), (4.17) and (4.18). In Example 4.2 on page 60 we demonstrated that these three big-M constraints differ in their tightness: (4.11) was tighter than (4.17), and (4.17) was tighter than (4.18). We also pointed out that greater tightness is preferred, as it brings the feasible region of the LP relaxation closer to the convex hull of integer-feasible solutions.

In the following, we consider only big-M constraints (4.11) and (4.17), and ignore the weakest Constraint set (4.18). Constraint set (4.11) requires $\sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst} \leq R_{kt} x_{kp}$ for each project $p \in \mathcal{P}$, each worker $k \in \mathcal{K}_p^{\text{suit}} \setminus \mathcal{K}_p^{\text{assigned}}$, and each period $t \in \mathcal{T}_p$. Constraint set (4.17) requires $y_{kpst} \leq R_{kt} x_{kp}$ for each project $p \in \mathcal{P}$, each worker $k \in \mathcal{K}_p^{\text{suit}} \setminus \mathcal{K}_p^{\text{assigned}}$, each skill $s \in \mathcal{S}_{kp}^{\text{match}}$, and each period $t \in \mathcal{T}_p$. In both (4.11) and (4.17), the availability R_{kt} of worker k limits her contribution to a project in period t.

Apart from the availability R_{kt}, there are two other limiting factors for the contribution of a worker k to a project p. These other two factors can be used to tighten the big-M constraints. The first additional limiting factor originates from the requirements r_{pst} for the skills $s \in \mathcal{S}_{kp}^{\text{match}}$ and from the corresponding skill levels l_{ks}. The requirements and the skill levels can limit the contribution of worker k to project p, as can be seen from the following example.

Example 6.5 Consider an instance where worker k and project p share only the skill s as a matching skill. Let $r_{pst} = 10$ be the requirement of project p for skill s in period t, let $R_{kt} = 100$ and $l_{ks} = 2$. Then, the contribution y_{kpst} of worker k to project p is limited to $\min\left(R_{kt}, \frac{r_{pst}}{l_{ks}}\right) = \min(100, 5) = 5$. □

The second limiting factor originates from the workload of department $d(k)$ to which worker k belongs and from the availabilities of her colleagues in department $d(k)$. The following example shows how the departmental workload and the availabilities can limit the contribution of worker k to project p.

Example 6.6 Consider an instance with three workers: $\mathcal{K} = \{k_1, k_2, k_3\}$. Let k_1 and k_2 belong to department d_1 and let k_3 be the sole worker of department d_2, i.e., $\mathcal{K}_{d_1} = \{k_1, k_2\}$ and $\mathcal{K}_{d_2} = \{k_3\}$. Let department d_1 have a requirement $rd_{d_1t} = 120$ in period t and let $rd_{d_2t} = 80$. Let $R_{kt} = 100$, $k \in \mathcal{K}$. Furthermore, let k_1, k_2, and k_3 master skill s, which is the only skill required by project p. Assume that $r_{pst} = 100$ in period t and that $l_{ks} = 1$, $k \in \mathcal{K}$. Then, the contribution y_{kpst} of each worker $k \in \mathcal{K}_{d_1}$ to project p is limited to $\min\left(R_{kt}, R_{kt} - \left(rd_{d(k)t} - \sum_{k' \in \mathcal{K}_{d(k)} \setminus \{k\}} R_{k't}\right)\right) = \min\left(R_{kt}, \left(\sum_{k' \in \mathcal{K}_{d(k)}} R_{k't}\right) - rd_{d(k)t}\right) = \min(100, 80) = 80$. Likewise, the contribution y_{kpst} of worker $k = k_3$ is limited to $\min(100, 20) = 20$. □

Hence, we can tighten big-M constraints (4.11) to

$$\sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst} \leq \min\left(R_{kt}, \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} \frac{r_{pst}}{l_{ks}}, \left(\sum_{k' \in \mathcal{K}_{d(k)}} R_{k't}\right) - rd_{d(k)t}\right) x_{kp} \quad \begin{array}{l} p \in \mathcal{P},\, t \in \mathcal{T}_p, \\ k \in \mathcal{K}_p^{\text{suit}} \setminus \mathcal{K}_p^{\text{assigned}} \end{array} \tag{6.5}$$

and big-M constraints (4.17) to

$$y_{kpst} \leq \min\left(R_{kt}, \frac{r_{pst}}{l_{ks}}, \left(\sum_{k' \in \mathcal{K}_{d(k)}} R_{k't}\right) - rd_{d(k)t}\right) x_{kp} \quad \begin{array}{l} p \in \mathcal{P},\, k \in \mathcal{K}_p^{\text{suit}} \setminus \mathcal{K}_p^{\text{assigned}}, \\ s \in \mathcal{S}_{kp}^{\text{match}},\, t \in \mathcal{T}_p \end{array} \tag{6.6}$$

In Subsection 4.3.1, we realized that big-M constraints (4.11) are tighter than big-M constraints (4.17). In general, the corresponding statement does not apply to (6.5) and (6.6). If $\frac{r_{pst}}{l_{ks}} \leq R_{kt}$ holds for at least one skill s, (6.6) can be tighter than (6.5).

For the network model, the big-M constraints can be tightened in the same way as for the standard model. Especially, big-M constraints (4.24) can be tightened analogously to (4.11).

Having outlined the three types of valid inequalities, we will briefly discuss the potential of these valid inequalities to act as cutting planes when the MIP model of the workforce assignment problem is solved by a branch-and-cut method. The globally valid inequalities that we devised for all three types of valid inequalities can be added to the model as constraints before the start of the branch-and-cut method. Since the total number of globally valid inequalities is not greater than $P \cdot S + P + K \cdot P \cdot T + K \cdot P \cdot S \cdot T$ and they replace around $K \cdot P \cdot T$ big-M constraints, the increase in the size of the model is moderate. Consequently, the globally valid inequalities can be added to the model and do not come into question as candidates for cutting planes.

The locally valid inequalities, which were devised only for the first and second type of valid inequalities, do not seem well suited for acting as cutting planes. This pessimistic statement holds for the case where we seek for a maximally violating cut among these locally valid inequalities in a node of a branch-and-cut tree as well as for the case where

we seek for an arbitrary cut among these valid inequalities. If a maximally violated valid inequality shall be found in a node of a branch-and-cut tree, it is necessary to compute all local lower bounds LB^{loc}_{ps}, $p \in \mathcal{P}$, $s \in \mathcal{S}_p$, and LB^{loc}_{p}, $p \in \mathcal{P}$. For each local lower bound, it must be checked to which extent the corresponding valid inequality is violated. If an arbitrary cut shall be found in a node, it is sufficient to compute local lower bounds one by one until the first one is encountered for which the corresponding valid inequality is violated, but in the worst case all local lower bounds and their valid inequalities are enumerated. There seems to be no obvious heuristic that quickly spots a violated inequality. Moreover, as we already indicated, the valid inequalities are rather weak. Hence, we suppose that integrating the locally valid inequalities into a branch-and-cut does not accelerate, but slow down the solution process. The conjecture will be checked in the numerical analysis in Subsection 7.3.1.

6.1.2 A polynomial-time algorithm for the utilization leveling problem

In this subsection, we will present a polynomial-time algorithm for the utilization leveling problem. The utilization leveling problem was described in Section 3.4 and modeled in Section 4.4. The objective of this problem is to allocate departmental workload in a period to department members such that their resulting working times are as equal as possible. The utilization leveling problem is tackled after project workload has been allocated to the workers of each department. One instance of this problem must be solved for each department $d \in \mathcal{D}$ and each period $t \in \mathcal{T}$. In Section 4.6, we argued that this problem is solvable in polynomial time (see Theorem 4.4 on page 86) but omitted a corresponding algorithm. Now, we will provide such an algorithm.

We developed an efficient algorithm for the utilization leveling problem, which we modeled in Section 4.4 as a linear program. We could apply a general-purpose method for linear programs in order to solve the leveling problem. The simplex method is such a general-purpose method. However, the simplex method has an exponential time complexity (cf. Goldfarb and Todd, 1989, pp. 130–133). There are other general-purpose methods, whose time complexity is polynomial, e.g., the ellipsoid method (cf. Chvátal, 1983, pp. 443–454; Neumann and Morlock, 2002, pp. 161–166) and interior point methods, also called projection methods (cf. Winston and Goldberg, 2004, pp. 190–191 and 597–605; Neumann and Morlock, 2002, pp. 166–171). However, as our leveling problem is quite simple, we developed a specially tailored algorithm that outperforms the aforementioned general-purpose methods with respect to worst-time complexity. Furthermore, for test instances our algorithm runs faster than the simplex method, which performs well in practice in spite of its exponential-time complexity (cf. Goldfarb and Todd, 1989, pp. 130–133).

We developed Algorithm 6.4 to solve the utilization leveling problem for a department d in a period t in polynomial time. Algorithm 6.4 levels the working times of all members of department d. The working time of a worker comprises the time that he spends for projects and the time in which he accomplishes departmental workload. The time that a worker spends for project work is already fixed at the point in time when the utilization leveling problem is considered. At this point in time, only the allocation of departmental workload can impact working times. We assume that departmental workload is arbitrarily

divisible, as discussed in Section 3.4. In Algorithm 6.4, the departmental workload rd_{dt} is allocated to the workers $k \in \mathcal{K}_d$ who belong to department d. For the allocation, we take into account the project workload of each worker $k \in \mathcal{K}_d$. We assign a departmental workload yd_{kt} to worker k for period t such that the working times of the department members are as equal as possible.

The basic idea of Algorithm 6.4 is to identify those workers in the department with the minimum working time. Then, departmental workload is steadily and equally allocated to those workers until one of the following three cases occurs: (1) no departmental workload remains for allocation, (2) the remaining availability of a worker becomes zero, (3) the working time of a worker reaches the level of another worker whose working time was above the minimum. This procedure of identifying workers with minimum working time and allocating departmental workload to these workers is repeated until no departmental workload is left for allocation.

At the outset of Algorithm 6.4, we initialize the remaining departmental workload rd_{dt}^{rem} that must be allocated with rd_{dt}. For each worker k in department d, we initialize his working time WT_{kt} in period t with the time he spends for project work, and his remaining availablity R_{kt}^{rem} with his remaining availability after project work (cf. line 8 of Algorithm 6.4, see also page 48). Next, all workers with positive remaining availability are identified and added to the set $\mathcal{K}^{\text{avail}}$ (cf. line 9).

As long as the departmental workload has not been completely allocated, the following steps are repeated. We determine the minimum working time $WT^{\min}$ among the working times of workers in $\mathcal{K}^{\text{avail}}$ and identify those workers whose working time is equal to the minimum working time $WT^{\min}$. We add those workers to the set $\mathcal{K}_{\min}^{\text{avail}}$. Given the set $\mathcal{K}_{\min}^{\text{avail}}$, we calculate an amount $depW$ of departmental workload that is allocated to each worker in $\mathcal{K}_{\min}^{\text{avail}}$.

For the calculation of the amount $depW$, we distinguish two different situations. Firstly, if all workers in $\mathcal{K}^{\text{avail}}$ have identical working times, i.e., if $\mathcal{K}_{\min}^{\text{avail}} = \mathcal{K}^{\text{avail}}$, the amount $depW$ is limited by the remaining departmental workload rd_{dt}^{rem} that must be allocated and by the smallest remaining availability among workers in $\mathcal{K}_{\min}^{\text{avail}}$ (cf. line 14). Secondly, if the working times of workers in $\mathcal{K}^{\text{avail}}$ are not identical, i.e., if $\mathcal{K}_{\min}^{\text{avail}} \subset \mathcal{K}^{\text{avail}}$, the amount $depW$ is additionally limited by the difference between the working time $WT^{\min}$ of workers in $\mathcal{K}_{\min}^{\text{avail}}$ and the next higher working time among workers in $\mathcal{K}^{\text{avail}}$ (cf. line 16).

After the departmental workload $depW$ was allocated to workers in $\mathcal{K}_{\min}^{\text{avail}}$, the remaining availability of some workers may have reached zero. These workers are removed from the set $\mathcal{K}^{\text{avail}}$ (cf. line 22).

When the complete departmental workload has been allocated, Algorithm 6.4 has found an optimal solution and terminates. Termination with an optimal solution is guaranteed because the existence of a feasible solution is secured and Algorithm 6.4 constructs a best solution among all feasible solutions. The existence of a non-empty set of feasible solutions is inherent in the solution of the workforce assignment problem which provides input data for the utilization leveling problem and assures that the departmental requirements can be accomplished.

Algorithm 6.4 runs in $O(|\mathcal{K}_d|^2)$ time assuming that the total time that each worker spends for project work is given. It levels the *absolute* working times of the department members. If we want to level the *relative* working times instead, only slight modifications

Algorithm 6.4 Algorithm to level working times within department d in period t by allocating departmental workload

Input: d; t; values of all variables y_{kpst} for period t, $k \in \mathcal{K}_d$; instance data
Output: Allocated departmental workloads yd_{kt}, $k \in \mathcal{K}_d$, for period t

$rd_{dt}^{\text{rem}} := rd_{dt}$;
for all $k \in \mathcal{K}_d$ **do**
    $yd_{kt} := 0$;
    $WT_{kt} := \sum_{p \in \mathcal{P}_k^{\text{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst}$;
    $R_{kt}^{\text{rem}} := R_{kt} - WT_{kt}$;
$\mathcal{K}^{\text{avail}} := \{k \in \mathcal{K}_d \mid R_{kt}^{\text{rem}} > 0\}$;
while $rd_{dt}^{\text{rem}} > 0$ **do**
    $WT^{\text{min}} := \min_{k \in \mathcal{K}^{\text{avail}}} WT_{kt}$;
    $\mathcal{K}_{\text{min}}^{\text{avail}} := \left\{k \in \mathcal{K}^{\text{avail}} \mid WT_{kt} = WT^{\text{min}}\right\}$;
    if $\mathcal{K}_{\text{min}}^{\text{avail}} = \mathcal{K}^{\text{avail}}$ **then**
        $depW := \min\left(\frac{rd_{dt}^{\text{rem}}}{|\mathcal{K}_{\text{min}}^{\text{avail}}|}, \min_{k \in \mathcal{K}_{\text{min}}^{\text{avail}}} R_{kt}^{\text{rem}}\right)$;
    else // $\mathcal{K}_{\text{min}}^{\text{avail}} \subset \mathcal{K}^{\text{avail}}$
        $depW := \min\left(\frac{rd_{dt}^{\text{rem}}}{|\mathcal{K}_{\text{min}}^{\text{avail}}|}, \min_{k \in \mathcal{K}_{\text{min}}^{\text{avail}}} R_{kt}^{\text{rem}}, \min_{k \in \mathcal{K}^{\text{avail}} \setminus \mathcal{K}_{\text{min}}^{\text{avail}}} WT_{kt} - WT^{\text{min}}\right)$;
    for all $k \in \mathcal{K}_{\text{min}}^{\text{avail}}$ **do**
        $yd_{kt} := yd_{kt} + depW$;
        $WT_{kt} := WT_{kt} + depW$;
        $R_{kt}^{\text{rem}} := R_{kt}^{\text{rem}} - depW$;
        if $R_{kt}^{\text{rem}} = 0$ **then**
            $\mathcal{K}^{\text{avail}} := \mathcal{K}^{\text{avail}} \setminus \{k\}$;
        $rd_{dt}^{\text{rem}} := rd_{dt}^{\text{rem}} - depW$;

of Algorithm 6.4 are necessary. Absolute and relative working times rely on the definitions of absolute and relative utilization in Section 4.4.

The worst-case time complexity of Algorithm 6.4 is clearly better than that of general-purpose methods for solving LPs. Let n denote the number of variables and m the number of constraints of an LP. While the simplex method has an exponential time complexity, the time complexity of the ellipsoid method is of order n^4 (cf. Goldfarb and Todd, 1989, pp. 138–140) and the time complexity of Karmarkar's projection method is of order $(m+n)^4$ (cf. Goldfarb and Todd, 1989, pp. 146–149). For the LP that we formulated for the utilization leveling problem, i.e., for model (4.39)–(4.44) on page 70, $n = |\mathcal{K}_d|^2$ holds. Hence, for the utilization leveling problem the time complexity of the ellipsoid method is of order $|\mathcal{K}_d|^8$, whereas Algorithm 6.4 runs in $O(|\mathcal{K}_d|^2)$ time.

6.2 Heuristic solution methods for the workforce assignment problem

In this section, we will outline four *heuristics* that we developed for the workforce assignment problem modeled in Subsection 4.3.1. Heuristics are solution methods that are, at least to some extent, specially tailored to a problem. Specially tailored means that heuristics exploit properties of a problem to find "good" solutions in acceptable time (cf. Aarts and Lenstra, 2003; Burke and Kendall, 2005, Chapters 4–8, 12–14, and 16–19; Neumann and Morlock, 2002, pp. 402–406; Domschke and Drexl, 2007, pp. 129–133). Usually, heuristics find good solutions, which may even be optimal. Though, heuristics for strongly NP-hard problems cannot guarantee to find arbitrarily good solutions in acceptable time. Some heuristics even cannot guarantee to find a feasible solution if the set of feasible solutions is non-empty.

Heuristics can be classified into *construction heuristics* and *improvement heuristics*. Construction heuristics systematically try to build a feasible solution for a problem. If a construction heuristic is deterministic, it can generate only one solution. If a construction heuristic entails stochastic decisions, it is randomized and can be embedded in a multi-start procedure to generate multiple solutions, from which a best solution can be picked. Improvement heuristics, in contrast to construction heuristics, cannot start from scratch. Improvement heuristics start with one or several solutions and try to derive better solutions.

We developed four construction heuristics. We also devised improvement heuristics, but so far we have not found a promising one. We tested a drop-add interchange heuristic and a genetic algorithm, but obtained disappointing results. In the following Subsections 6.2.1–6.2.4, we present our four construction heuristics, each of them can be embedded in a multi-start procedure.

In Subsection 6.2.1, we outline a greedy randomized assignment procedure, called GRAP. GRAP is similar to what a manual solution procedure may look like: Project by project is selected; workers are assigned to the currently selected project and project workload is allocated to the workers until all workload of the project is covered. Then, the next project is considered. We outline also a modified version of GRAP, which we term ModGRAP.

In Subsection 6.2.2, we describe an iterated simultaneous assignment procedure, called ISAP, which is an advanced version of GRAP. ISAP assigns in each iteration to each project a worker such that no worker is assigned to more than one project. The assignment is based on a perfect matching of maximum suitability between projects and workers, where dummy projects or dummy workers ensure the existence of a perfect matching.

Subsection 6.2.3 deals with the third construction heuristic, which is a drop procedure where at the outset each worker is assigned to every suitable project and then assignments are dropped, i.e., cancelled, one by one until any further cancellation would lead to infeasibility. This procedure is called DROP.

Finally, a rounding heuristic called ROUND is unrolled in Subsection 6.2.4. ROUND iteratively solves the linear programming relaxation of the workforce assignment problem. In each iteration, some binary variables with fractional values are rounded up and fixed to 1 until an integer-feasible solution is reached. In Subsection 6.2.4, we also sketch a

relax-and-fix approach. Since the performance of this approach was disappointing, the approach is abandoned thereafter.

Each solution method for the workforce assignment problem must determine values for the variables $\boldsymbol{x}$ and $\boldsymbol{y}$. The four heuristics GRAP, ISAP, DROP, and ROUND differ from one another in the extent to which a solver is used to determine values for $\boldsymbol{x}$ and $\boldsymbol{y}$. Here, the term *solver* means any exact optimization algorithm. For example, the LP solver and the MIP solver of CPLEX are exact optimization algorithms. If we apply the MIP solver of CPLEX to optimally solve an instance of the workforce assignment problem, the branch-and-cut method of CPLEX determines optimal values for $\boldsymbol{x}$ and $\boldsymbol{y}$. This exact method can be considered an extreme in the spectrum of our solution methods. The extreme on the opposite side is the heuristic GRAP, which determines values for $\boldsymbol{x}$ and $\boldsymbol{y}$ in a greedy, myopic fashion without a solver.

Table 6.1 summarizes how the solution methods determine values for the variables $\boldsymbol{x}$ and $\boldsymbol{y}$. The entry "CPLEX" in Table 6.1 refers to the branch-and-cut method of the solver package CPLEX and represents an exact method, whereas the other entries represent heuristic methods. We listed the solution methods in order of increasing solver usage. While GRAP does not use a solver at all, ISAP applies a solver to determine values for $\boldsymbol{x}$. Though, it is not guaranteed that globally optimal values for the variables $\boldsymbol{x}$ are determined, because in each iteration of ISAP the solver is only used to greedily expand a partial solution. Such a greediness is documented in the column headed "rigor" for each variable type in Table 6.1. In regard to variables $\boldsymbol{y}$, the rigor is exact if the respective solution method finds an existing feasible solution for the variables $\boldsymbol{y}$ given values for the variables $\boldsymbol{x}$. The rigor is greedy if the solution method can fail to find an existing feasible solution for the variables $\boldsymbol{y}$. Among the heuristics, ROUND exhibits the greatest solver usage.

Table 6.1: Solution methods for the workforce assignment problem and characteristics of their way to determine values for the variables $\boldsymbol{x}$ and $\boldsymbol{y}$

	$\boldsymbol{x}$		$\boldsymbol{y}$	
	solver	rigor	solver	rigor
GRAP	–	greedy	–	greedy
ISAP	✓	greedy	–	greedy
DROP	–	greedy	✓	exact
ROUND (and relax-and-fix)	✓	greedy	✓	exact
CPLEX (branch-and-cut)	✓	exact	✓	exact

With respect to the determination of variables $\boldsymbol{x}$, the heuristics GRAP, ISAP, and ROUND are similar, as they start from a point with no assignments and then greedily assign workers to projects. However, the assignments of ROUND are less greedy than those of GRAP and ISAP, because ROUND exploits information from the LP relaxation to decide about every assignment. The philosophy of DROP is contrary to that of GRAP, ISAP, and ROUND, because DROP starts with all possible assignments and tries to cancel these assignments one by one.

With respect to the determination of variables $\boldsymbol{y}$, two different ways are followed by the heuristics. On the one hand, GRAP and ISAP greedily allocate project workload for

each assignment of a worker k to a project p. For an assignment (k, p), both heuristics allocate as much workload of project p as possible to worker k and this allocation is fixed. On the other hand, there are DROP and ROUND, where an allocation of project workload is never fixed during the respective procedure even if the corresponding variable x_{kp} has already been fixed to 1. Until the very end of the procedure, the allocation can be changed in order to adjust the allocation of project workload to changes in the values of the variables $\boldsymbol{x}$ that have not been fixed so far.

GRAP and ISAP are construction heuristics which may fail to find a feasible solution, but which are very fast. DROP and ROUND, in contrast, are construction heuristics that guarantee to find a feasible solution if any exists. DROP can also be used to seek for better solutions in the neighborhood of a given solution, but it would go too far to classify DROP as an improvement heuristic. Especially in case of instances characterized by high workforce utilization, GRAP and ISAP often fail to find an existing feasible solution, which is always found by DROP and ROUND. Although the computation times of DROP and ROUND are not negligible, these are methods that can find a feasible solution for larger-sized instances in acceptable time, whereas branch-and-cut methods frequently fail to find a feasible solution within acceptable time.

We embedded each of our four heuristics in a multi-start procedure. For each heuristic, the multi-start procedure is the same in principle and follows the randomized multi-start algorithm outlined by Martí et al. (2013, p. 3, Algorithm 4). Multi-start procedures are also called multi-pass procedures. In every pass, it is tried to construct a solution. The best solution constructed is returned.

Algorithm 6.5 summarizes our multi-start procedure, which applies a heuristic repeatedly in order to construct several solutions. Randomness is incorporated by stochastic choices during construction of a solution. If a solution is better than every solution constructed so far, this solution is stored as the incumbent solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$. The incumbent solution is at any point in time the best integer-feasible solution found until this point in time. Solutions are constructed until a stopping criterion is satisfied. Several stopping criteria can be applied simultaneously, e.g., a time limit and a limit on the number of passes, which limits the number of solutions that are constructed.

Algorithm 6.5 Multi-start procedure for heuristics for the workforce assignment problem

Input: Instance data, heuristic $H \in \{\text{GRAP, ISAP, DROP, ROUND}\}$
Output: A feasible solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ for the workforce assignment problem and its objective function value f^* or no solution

$f^* := \sum_{k \in K} \left|\mathcal{P}_k^{\text{suit}}\right| + 1$; // Initialize f^* with the worst-case value increased by 1
$(\boldsymbol{x}^*, \boldsymbol{y}^*) := (\boldsymbol{0}, \boldsymbol{0})$; // Mark $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ initially as infeasible
while none of the stopping criteria is met **do**
    Construct a random solution $(\boldsymbol{x}, \boldsymbol{y})$ with heuristic H;
    if $(\boldsymbol{x}, \boldsymbol{y})$ is feasible **then**
        if $f((\boldsymbol{x}, \boldsymbol{y})) < f^*$ **then**
            $f^* := f((\boldsymbol{x}, \boldsymbol{y}))$;
            $(\boldsymbol{x}^*, \boldsymbol{y}^*) := (\boldsymbol{x}, \boldsymbol{y})$;

Our multi-start procedure can be classified as memory-less, randomized, and build-

from-scratch according to the classification scheme of Martí et al. (2010, pp. 273–274). They identified three key attributes of multi-start methods, namely, (1) *memory*, (2) *randomization*, and (3) *degree of rebuild*. Each attribute ranges between two poles. These poles are (1) memory-less vs. memory-based, (2) randomized vs. systematic, and (3) build-from-scratch vs. rebuild, respectively. A memory-based multi-start method can, for example, store and exploit information about common characteristics of the best solutions that have been constructed so far. A systematic multi-start method would ensure a minimum level of diversity in solutions that are constructed, while a randomized method, which also aims at diversity, cannot guarantee a prespecified level of diversity. The third attribute, namely, the degree of rebuild, measures the extent to which variables are fixed to values they took on in solutions of previous passes. The two poles of this attribute are *build-from scratch*, where no variables are fixed, and *rebuild*, where all variables are fixed to values derived from known solutions. Our methods DROP and ROUND actually exhibit a small degree of rebuild. For every heuristic that we describe in the following subsections, we will state in more detail how the multi-start procedure was implemented.

To give a concise description of the four heuristics in the following subsections, we cut our presentation short at two points. First, we will present pseudo code only for the heuristics themselves and omit all overhead that is required for a multi-start version. The essence of this overhead was presented in Algorithm 6.5. Nevertheless, we describe how the multi-start procedure was implemented. Second, we assume that there are no workers who have already been assigned to an (ongoing) project, in other words, we assume that $\mathcal{K}_p^{\text{assigned}} = \emptyset$ holds for all $p \in \mathcal{P}$. Without much effort, our heuristics can be extended to the case where $\mathcal{K}_p^{\text{assigned}} \neq \emptyset$ holds for all or some $p \in \mathcal{P}$. In fact, we developed and implemented our heuristics to deal with the situation where workers have already been assigned to projects.

6.2.1 A greedy randomized assignment procedure (GRAP)

In this subsection, we will present a construction heuristic that we call *greedy randomized assignment procedure* (GRAP). Its basic idea is to select for each project p worker by worker and to allocate as much workload of project p to each worker until all workload of project p is covered. The heuristic is called *greedy randomized*, because when decisions are made, e.g., about the worker that is assigned to a project next, we randomly select an alternative with a certain probability. This probability is proportional to a local or myopic attractiveness of the alternative. Randomness is incorporated in order to facilitate an extension of this heuristic to a multi-start method.

At the beginning of this subsection, we will give a short overview of the heuristic and explain the idea behind. Then, we will briefly judge the relevance of this heuristic for firms and for our work. After that, we will describe the heuristic in detail and provide two algorithms that illuminate essential calculations of the GRAP method. A multi-start version of GRAP will be explained in brief. Eventually, we will discuss the drawback that GRAP can fail to find an existing feasible solution. We outline and discuss a modified version of GRAP that overcomes the drawback but yields solutions of minor quality than GRAP on average.

In short, GRAP works as follows. It starts with an initially infeasible solution where neither any departmental nor any project workload has been allocated and where no

worker is assigned to any project. Project by project is randomly selected and for each project p worker by worker is randomly selected to cover requirements of project p. For each worker k that is selected for project p, matching skill by matching skill is randomly selected. For each selected matching skill, as much workload of this skill as possible is allocated to worker k until either the workload of all matching skills is allocated or until worker k cannot cover any more workload of project p. In this case, the next worker is selected for project p. When all workload of project p is covered, the next project is selected.

This procedure of extending a partial solution, i.e., the procedure of selecting a project, selecting workers, and allocating project workload is repeated until a feasible solution is constructed, in which all projects are staffed, or until a point is reached where it is detected that the partial solution cannot be extended to a feasible solution. This point is reached when there is a project with uncovered workload and no suitable worker has sufficient remaining availability to accomplish this workload. In this case, GRAP has failed to construct a feasible solution due to a misallocation of workload in the course of constructing a solution.

A procedure similar to GRAP is used by Gutjahr et al. (2008, pp. 293–294) to assign project workloads that require different skills to multi-skilled workers. However, their procedure is not randomized and not aimed at minimizing the number of assignments. Their objective for the procedure is merely to quickly find a feasible allocation of workload. They consider project by project in order of non-decreasing project due dates, and for each project they consider skill by skill in order of non-increasing workload. For allocating workload of each skill, the workers are considered in order of non-increasing skill levels. This procedure tends to lead to a large number of assignments, but this tendency does not contradict their objective to find a feasible allocation of workload. Yet, the procedure of Gutjahr et al. (2008) can fail to find an existing feasible solution like GRAP can.

The idea behind GRAP is the assumption that in good solutions, i.e., in solutions with a small number of assignments, a worker contributes as much as possible to each project he is assigned to. From the myopic view of GRAP, it seems advantageous if a worker k spends his entire time for one project p, because this concentration of worker k on project p results in only one assignment for worker k. At the same time, for project p, the number of required additional workers tends to be as small as possible if worker k accomplishes as much workload as possible. This local advantage of assigning as much workload as possible to a worker was our motivation for GRAP. However, this local advantage paired with the strategy of sequential assignments may be globally disadvantageous and lead to solutions of low quality.

In our opinion, a heuristic like GRAP is of interest for both practice and theory, in spite of being a rather simple heuristic that may provide solutions of minor quality. In practice, many firms are confronted with some variant of our workforce assignment problem. Planners in these firms who want to form small teams are likely to apply a procedure similar to GRAP in order to determine a staffing plan and to assign work packages to workers. Hence, our heuristic GRAP mimics a procedure that a planner may intuitively follow if he has to figure out a solution to his workforce assignment problem and lacks other methods. If the planner plans manually, then GRAP will be a great support for the planner and save him time. From a theoretical point of view, the GRAP heuristic can be used to determine a minimum standard with respect to solution quality.

If another method does not meet the solution quality of GRAP, it is most likely of no interest for a firm.

In the subsequent paragraphs, we will describe the GRAP heuristic in detail. We will divide our description into two parts. The first part deals with preparatory initializations, the second part describes the main part of GRAP.

The first part of GRAP is outlined in Algorithm 6.6. At the beginning of the GRAP procedure, a number of declarations and initializations are necessary. We have to define sets and variables that we need in order to track remaining unstaffed projects, remaining availabilities of workers, remaining requirements of projects, and so on. Algorithm 6.6 summarizes all these preparatory declarations and initializations.

The initializations start by setting all variables x_{kp} and y_{kpst} to 0 (cf. line 4 of Algorithm 6.6). The list $\mathcal{P}^{\text{toStaff}}$ that contains all remaining unstaffed projects is initialized with all projects $p \in \mathcal{P}$.

For each project $p \in \mathcal{P}$, we define the set $\mathcal{S}_p^{\text{rem}}$ that contains all skills which are required by project p and for which uncovered workload exists. Additionally, we initialize the set of remaining suitable workers $\mathcal{K}_p^{\text{suit,rem}}$ with the set $\mathcal{K}_p^{\text{suit}}$ of all workers that are suitable for project p. For each worker k in the set $\mathcal{K}_p^{\text{suit}}$, we define the set $\mathcal{S}_{kp}^{\text{match,rem}}$ that includes all matching skills between k and p for which uncovered remaining requirements exist. For each period t of execution of project p and for each required skill s of project p, we initialize the variable r_{pst}^{rem} that records the residual requirement of project p for skill s in period t.

Algorithm 6.6 Initializations for the heuristic GRAP

1: **Input:** Instance data
2: **Output:** Data required by the main part of the heuristic GRAP
3: ---
4: $\boldsymbol{x} := \mathbf{0}, \quad \boldsymbol{y} := \mathbf{0}$;
5: $\mathcal{P}^{\text{toStaff}} := \mathcal{P}$;
6: **for all** $p \in \mathcal{P}$ **do**
7: $\quad \mathcal{S}_p^{\text{rem}} := \mathcal{S}_p$;
8: $\quad \mathcal{K}_p^{\text{suit,rem}} := \mathcal{K}_p^{\text{suit}}$;
9: $\quad$ **for all** $k \in \mathcal{K}_p^{\text{suit}}$ **do**
10: $\quad\quad \mathcal{S}_{kp}^{\text{match,rem}} := \mathcal{S}_{kp}^{\text{match}}$;
11: $\quad$ **for all** $t \in \mathcal{T}_p$ **do**
12: $\quad\quad$ **for all** $s \in \mathcal{S}_p$ **do**
13: $\quad\quad\quad r_{pst}^{\text{rem}} := r_{pst}$;
14: **for all** $t \in \mathcal{T}$ **do**
15: $\quad$ **for all** $k \in \mathcal{K}$ **do**
16: $\quad\quad R_{kt}^{\text{rem}} := R_{kt}$;
17: $\quad$ **for all** $d \in \mathcal{D}$ **do**
18: $\quad\quad totR_{dt}^{\text{rem}} := \sum_{k \in \mathcal{K}_d} R_{kt}$;

Furthermore, we define for each period $t \in T$ two additional types of variables that are required to track residual quantities. The first type of variables is denoted by R_{kt}^{rem}, $k \in \mathcal{K}$. Variable R_{kt}^{rem} records for worker k her remaining availability in period t. The

second type of variables is denoted by $totR^{\text{rem}}_{dt}$, $d \in \mathcal{D}$. Variable $totR^{\text{rem}}_{dt}$ records the total residual availability of all members of department d in period t. We need this last type of variables to ensure that the departmental requirements can be satisfied at any time during the GRAP procedure. Algorithm 6.6 requires $O(KPST + TDK)$ time.

The main part of the GRAP procedure is summarized in Algorithm 6.7 which tries to construct a feasible solution for the workforce assignment problem. Algorithm 6.7 can be broken down into four parts: a selection part (lines 4–11 of Algorithm 6.7), an allocation part (lines 12–14), a part for updates of variables (lines 15–21), and a part for updates of sets (lines 22–34).

In the selection part, the construction of a solution starts with the non-empty list $\mathcal{P}^{\text{toStaff}}$, which contains the projects that must be staffed. We randomly select a project from this list, where the selection probability is the same for each project. Of course, other selection probabilities can be assigned to the projects. For example, one could bias the selection towards projects with a small number of remaining suitable workers or towards projects with a high number of required skills.

After selection of a project p and as long as the selected project p has skills with uncovered remaining requirements, we randomly select a worker k in order to allocate workload of project p to worker k. If no suitable worker is left for project p, a feasible solution cannot be reached and the GRAP procedure is terminated. Otherwise, we prepare the selection of a worker by calculating a non-negative suitability value $suit_{kp}$ for each worker $k \in \mathcal{K}^{\text{suit,rem}}_{p}$ who can be assigned to project p. Then, we apply a roulette wheel selection where the probability for the selection of worker $k \in \mathcal{K}^{\text{suit,rem}}_{p}$ is proportional to $suit_{kp}$. How we actually calculate the values $suit_{kp}$ will be explained after the overview of Algorithm 6.7.

After we selected worker k for project p, a skill s from the set of remaining matching skills between k and p is chosen, as long as this set is not empty. From the remaining requirement of project p for skill s, we will allocate as much as possible to worker k. But let us first explain the selection of a skill in more detail. For the selection of a skill, we consider only the "best skills", which are those skills in $\mathcal{S}^{\text{match,rem}}_{kp}$ that have the highest skill level among all the skills in $\mathcal{S}^{\text{match,rem}}_{kp}$. To each of these "best skills", we assign the same selection probability. Then, one skill is chosen randomly. For example, let s_1, s_2, and s_3 be the remaining matching skills between worker k and project p, and let $l_{ks_1} = 1.5$, $l_{ks_2} = 1$, and $l_{ks_3} = 1.5$ be the respective skill levels of worker k. We select s_1 or s_3, both with a probability of 0.5. This greedy and myopic selection procedure for a skill outperformed several other simple selection procedures in preliminary tests.

After we selected skill s, the allocation part of GRAP starts. We allocate for each period $t \in \mathcal{T}_p$ as much of the remaining requirement r^{rem}_{pst} as possible to worker k (cf. line 14).

When workload has been allocated, the part for updates of variables begins. We update the variables that track remaining quantities and use the variable *totSkillContr* to store the total time that worker k contributes to skill s over all periods $t \in \mathcal{T}_p$. If workload was allocated to worker k, *totSkillContr* is positive and we assign worker k to project p by setting $x_{kp} := 1$. Furthermore, we calculate the total remaining requirement $totReq^{\text{rem}}_{ps}$ of project p for skill s over all periods $t \in \mathcal{T}_p$. This variable $totReq^{\text{rem}}_{ps}$ is required for the following update of sets.

The last part of Algorithm 6.7 deals with updating sets that are necessary to track

Algorithm 6.7 Main part of the heuristic GRAP

Input: Instance data, output of Algorithm 6.6
Output: A feasible solution $(\boldsymbol{x}, \boldsymbol{y})$ for the workforce assignment problem or no solution

---

while $\mathcal{P}^{\text{toStaff}} \neq \emptyset$ **do**
  Randomly select a project p from the list $\mathcal{P}^{\text{toStaff}}$;
  while $\mathcal{S}_p^{\text{rem}} \neq \emptyset$ **do**
    if $\mathcal{K}_p^{\text{suit,rem}} = \emptyset$ **then**
      terminate; // GRAP failed to construct a feasible solution
    Randomly select a worker k from the set $\mathcal{K}_p^{\text{suit,rem}}$;
    while $\mathcal{S}_{kp}^{\text{match,rem}} \neq \emptyset$ **do**
      Randomly select a skill s from the set $\mathcal{S}_{kp}^{\text{match,rem}}$;
      $totSkillContr := 0$;
      for all $t \in \mathcal{T}_p$ **do**
        $y_{kpst} := \min\left(\frac{r_{pst}^{\text{rem}}}{l_{ks}},\ R_{kt}^{\text{rem}},\ totR_{dt}^{\text{rem}} - rd_{dt}\right)$;
        $r_{pst}^{\text{rem}} := r_{pst}^{\text{rem}} - l_{ks} y_{kpst}$;
        $R_{kt}^{\text{rem}} := R_{kt}^{\text{rem}} - y_{kpst}$;
        $totR_{dt}^{\text{rem}} := totR_{dt}^{\text{rem}} - y_{kpst}$;
        $totSkillContr := totSkillContr + y_{kpst}$;
      $totReq_{ps}^{\text{rem}} := \sum_{t \in \mathcal{T}_p} r_{pst}^{\text{rem}}$;
      if $totSkillContr > 0$ **then**
        $x_{kp} := 1$;
      $\mathcal{S}_{kp}^{\text{match,rem}} := \mathcal{S}_{kp}^{\text{match,rem}} \setminus \{s\}$;
      if $\mathcal{S}_{kp}^{\text{match,rem}} = \emptyset$ **then**
        $\mathcal{K}_p^{\text{suit,rem}} := \mathcal{K}_p^{\text{suit,rem}} \setminus \{k\}$;
        if $totReq_{ps}^{\text{rem}} > 0$ **then**
          break; // Abort the while-loop started in line 10, i.e., select another worker
      if $totReq_{ps}^{\text{rem}} = 0$ **then**
        for all $k' \in \left(\mathcal{K}_p^{\text{suit,rem}} \cap \mathcal{K}_s\right) \mid k' \neq k$ **do**
          $\mathcal{S}_{k'p}^{\text{match,rem}} := \mathcal{S}_{k'p}^{\text{match,rem}} \setminus \{s\}$;
          if $\mathcal{S}_{k'p}^{\text{match,rem}} = \emptyset$ **then**
            $\mathcal{K}_p^{\text{suit,rem}} := \mathcal{K}_p^{\text{suit,rem}} \setminus \{k'\}$;
        $\mathcal{S}_p^{\text{rem}} := \mathcal{S}_p^{\text{rem}} \setminus \{s\}$;
        if $\mathcal{S}_p^{\text{rem}} = \emptyset$ **then**
          $\mathcal{P}^{\text{toStaff}} := \mathcal{P}^{\text{toStaff}} \setminus \{p\}$;

---

remaining matching skills, remaining suitable workers, and so on. This part starts with removing skill s from the set $\mathcal{S}_{kp}^{\text{match,rem}}$ of remaining matching skills between worker k and project p. Skill s can be removed, because the maximum possible workload of skill s was allocated to worker k. If skill s was the only remaining matching skill, worker k

is removed from the set $\mathcal{K}_p^{\text{suit,rem}}$ of remaining suitable workers for project p, because worker k cannot make any further contribution to project p. If worker k cannot make any further contribution and if project p has positive remaining requirements for skill s, i.e., if $totReq_{ps}^{\text{rem}}$ is positive, another worker must be selected to whom workload can be allocated.

If project p has no remaining requirements for skill s, because worker k could cover all remaining requirements of project p for skill s, we make two updates. First, we remove skill s from the sets of remaining matching skills of all other workers k' who may be able to cover requirements of project p for skill s. If skill s was the only remaining matching skill between worker k' and project p, we remove worker k' from the set of remaining suitable workers for project p. Second, we remove skill s from the set of remaining skills which are required by project p and for which uncovered requirements exist. If skill s was the last skill with uncovered requirements, we remove project p from the list $\mathcal{P}^{\text{toStaff}}$, because all requirements of project p are covered.

After the updates, GRAP continues in one of three ways. The first way is that another remaining matching skill between worker k and project p is selected. The second way is that another worker for project p is selected. The third way is that another project is selected for staffing if there is any left. Algorithm 6.7 runs in $O(PK^2ST)$ time, and hence, GRAP requires $O(PK^2ST)$ time because $D \leq K$ holds.

Having given an overview of GRAP, we come back to the suitability values $suit_{kp}$, $k \in \mathcal{K}_p^{\text{suit,rem}}$, which are calculated before a worker $k \in \mathcal{K}_p^{\text{suit,rem}}$ is selected for a project $p \in \mathcal{P}^{\text{toStaff}}$. Manifold possibilities exist to calculate the values $suit_{kp}$. We will suggest two possibilities.

Our first way to calculate the value $suit_{kp}$ considers the sum of the skill levels that are associated with the remaining matching skills between worker k and project p. Additionally, it considers the number of projects to which worker k is already assigned. We denote this suitability value by $suit_{kp}^{\text{A}}$, $p \in \mathcal{P}^{\text{toStaff}}$; its calculation is given in Definitions (6.7).

$$suit_{kp}^{\text{A}} := \frac{\sum\limits_{s \in \mathcal{S}_{kp}^{\text{match,rem}}} l_{ks}}{1 + \sum\limits_{p' \in \mathcal{P}_k^{\text{suit}}} x_{kp'}} \qquad k \in \mathcal{K}_p^{\text{suit,rem}} \tag{6.7}$$

In case of suitability values $suit_{kp}^{\text{A}}$, the selection probability of worker k is the greater the greater the sum of the skill levels that are associated with the remaining matching skills between worker k and project p. And, the probability is the greater the lower the number of projects to which worker k has already been assigned. The probability of selecting worker k for project p is dynamic and hence can change in the course of the GRAP procedure.

Definitions (6.7) ignore the remaining availabilities of worker k as well as the extent of remaining requirements of project p measured in man-hours. Though, remaining availabilities and remaining requirements seem to be a reasonable indicator for the suitability of a worker for a project. That is why these indicators are taken into account by our second way to calculate the value $suit_{kp}$.

Our second way regards the remaining availabilities of worker k and the man-hours that are required by project p but that have not been allocated so far. The resulting suitability value is deonoted by $suit_{kp}^{\text{B}}$. The computation of $suit_{kp}^{\text{B}}$ is given in Algorithm 6.8.

Here, $suit^{\mathrm{B}}_{kp}$ is defined as the share of all remaining requirements of project p that can be covered by worker k. Algorithm 6.8 computes for each period $t \in \mathcal{T}_p$ the maximum number of man-hours of the remaining matching requirements that can be accomplished by worker k and cumulates these man-hours over all periods of project duration. At the end, the cumulated man-hours are divided by the sum of all remaining requirements of project p to obtain the share that worker k can cover (cf. line 13 of Algorithm 6.8).

Algorithm 6.8 Calculation of the suitability value $suit^{\mathrm{B}}_{kp}$

1: **Input:** Project $p \in \mathcal{P}^{\mathrm{toStaff}}$, worker $k \in \mathcal{K}^{\mathrm{suit,rem}}_{p}$
2: **Output:** Suitability value $suit^{\mathrm{B}}_{kp}$
3: ---
4: $maxHours := 0$
5: **for all** $t \in \mathcal{T}_p$ **do**
6: $\quad avail^{\mathrm{rem}} := \min\left(R^{\mathrm{rem}}_{kt},\ totR^{\mathrm{rem}}_{d(k)t} - rd_{d(k)t}\right)$;
7: $\quad$ **for all** $s \in \mathcal{S}^{\mathrm{match,rem}}_{kp}$ in order of non-increasing skill levels l_{ks} **do**
8: $\quad\quad maxHoursSkill := \min\left(r^{\mathrm{rem}}_{pst},\ avail^{\mathrm{rem}} l_{ks}\right)$;
9: $\quad\quad maxHours := maxHours + maxHoursSkill$;
10: $\quad\quad avail^{\mathrm{rem}} := avail^{\mathrm{rem}} - \frac{maxHoursSkill}{l_{ks}}$;
11: $\quad\quad$ **if** $avail^{\mathrm{rem}} = 0$ **then**
12: $\quad\quad\quad$ **break**; // Abort the for-loop over the skills $s \in \mathcal{S}^{\mathrm{match,rem}}_{kp}$
13: $suit^{\mathrm{B}}_{kp} := \frac{maxHours}{\sum_{s \in \mathcal{S}^{\mathrm{rem}}_{p}} \sum_{t \in \mathcal{T}_p} r^{\mathrm{rem}}_{pst}}$;

Let us explain in more detail how we determine the maximum number of required man-hours of project p that worker k can cover. This maximum number of man-hours is denoted by $maxHours$. It is the sum of the maximum numbers of man-hours that worker k can accomplish in the periods $t \in \mathcal{T}_p$. To obtain the maximum contribution of worker k in a period $t \in \mathcal{T}_p$, we first determine the remaining availability $avail^{\mathrm{rem}}$ of worker k (cf. line 6). This remaining availability takes into account that worker k must accomplish departmental workload in cases where the remaining availabilities $R^{\mathrm{rem}}_{k't}$ of all other department members k' are not sufficiently large to fully cover the departmental requirement. To take into account the requirement rd_{dt} of the department d to which worker k belongs, the variable $totR^{\mathrm{rem}}_{dt}$ is considered. This variable tracks the total remaining availability of all department members and was introduced on page 130.

Given the remaining availability $avail^{\mathrm{rem}}$ of worker k, we determine the maximum number of required man-hours of project p that worker k can cover in period t. We iterate over all remaining matching skills between k and p in order of non-increasing skill levels. For a remaining matching skill s, we calculate the maximum number of man-hours of the remaining requirement r^{rem}_{pst} that worker k can cover, store it in the variable $maxHoursSkill$, add it to the variable $maxHours$, update the remaining availability $avail^{\mathrm{rem}}$ of worker k, and continue with the next skill. The calculation of the maximum contribution of worker k in period t stops if the remaining availability $avail^{\mathrm{rem}}$ of worker k is depleted. This calculation is done for each period. Finally, the cumulated number of man-hours that worker k can accomplish is divided by the total number of remaining required man-hours that include also remaining requirements for skills that do not belong to the matching

skills of k and p. Algorithm 6.8 requires $O(S^2 + TS)$ time if quicksort is used for sorting operations.

For a multi-start version, we directly embedded GRAP in the generic multi-start procedure outlined in Algorithm 6.5 on page 126. We directly inserted Algorithms 6.6 and 6.7 in Algorithm 6.5. This multi-start version of GRAP constructs different solutions, because projects, workers, and skills are randomly selected within GRAP leading to different assignments and workload allocations.

Having outlined the GRAP procedure and its multi-start version, we will turn to an important drawback of GRAP. The drawback is that there are instances where GRAP is unable to find an existing optimal or feasible solution even if GRAP is executed an infinite number of times. Even if all possible orders are considered in which projects, workers, and skills can be selected by GRAP, GRAP can fail to construct an existing optimal or feasible solution. Though, this drawback can be overcome by a modified version of GRAP that we call ModGRAP. The heuristic ModGRAP has another drawback, however. Its expected average solution quality is worse than the solution quality of GRAP.

In the following, we will prove the unpleasant property that GRAP can miss an existing optimal or feasible solution. Then we will present the modified version ModGRAP and prove that it cures the drawback of GRAP. Finally, we will suggest that this cure comes at a high cost, as solution quality suffers.

Proposition 6.1 *The heuristic GRAP can fail to find an existing optimal or feasible solution even if GRAP is applied repeatedly in order to select projects, workers, and skills in all possible orders.* □

Proof For our proof, we will present an instance for which GRAP cannot construct a feasible solution. This instance features two workers k_1 and k_2, two skills s_1 and s_2, and two projects p_1 and p_2, which last only for one period t. Both projects require both skills. The requirements are identical: $r_{p_1 s_1 t} = r_{p_1 s_2 t} = r_{p_2 s_1 t} = r_{p_2 s_2 t} = 1$. Both workers master both skills with contrary skill levels: $l_{k_1 s_1} = 2$, $l_{k_1 s_2} = 0.5$, $l_{k_2 s_1} = 0.5$, $l_{k_2 s_2} = 2$. While worker k_1 is very experienced in skill s_1 and a beginner in skill s_2, the opposite holds for worker k_2. Let $R_{kt} = 1$, $k \in \{k_1, k_2\}$. There is no departmental workload.

Assume that GRAP selects project p_1 and worker k_1 first. Then, the corresponding allocation outcome is $y_{k_1 p_1 s_1 t} = 0.5$ and $y_{k_1 p_1 s_2 t} = 0.5$. Worker k_1 cannot contribute to any further project, because her remaining availability is zero. Now, skill s_2 is the only skill of project p_1 with a positive remaining requirement, this remaining requirement amounts to $r^{\text{rem}}_{p_1 s_2 t} = 0.75$. This remaining requirement is completely allocated to worker k_2 resulting in $y_{k_2 p_1 s_2 t} = 0.375$. But now the remaining availability $R^{\text{rem}}_{k_2 t} = 0.625$ of worker k_2 is not large enough to cover the requirements of project p_2. Only a remaining requirement $R_{k_2 t} \geq 2.5$ would be large enough. Hence, a feasible solution cannot be constructed.

Every other selection of a project and a worker at the beginning of GRAP leads to the same situation of a gap between required and available working time. GRAP ends up at a point where at most three assignments can be made, whereas every feasible solution to this instance requires four assignments, namely, $x_{kp} = 1$, $k \in \{k_1, k_2\}$, $p \in \{p_1, p_2\}$. The sole feasible and hence optimal allocation of workload is $y_{k_1 p_1 s_1 t} = y_{k_1 p_2 s_1 t} = 0.5$ for worker k_1 and $y_{k_2 p_1 s_2 t} = y_{k_2 p_2 s_2 t} = 0.5$ for worker k_2. ■

Algorithm 6.9 sketches a modified version of GRAP that can find a feasible solution to the instance that we described in the proof of Proposition 6.1. The modified version of

GRAP, called ModGRAP, randomly chooses a project $p \in \mathcal{P}^{\text{toStaff}}$, a worker $k \in \mathcal{K}_p^{\text{suit,rem}}$, a skill $s \in \mathcal{K}_p^{\text{suit,rem}}$, and a period $t \in \mathcal{T}_p$ with a positive remaining requirement r_{pst}^{rem}. As much of the remaining requirement r_{pst}^{rem} as possible is allocated to worker k. Updates of variables including the variable x_{kp} and updates of sets are executed similarly to GRAP. Then, the next quadruplet (p, k, s, t) is randomly determined and for the corresponding variable y_{kpst} the maximum value is calculated. This procedure is repeated as long as projects with positive remaining requirements exist, i.e., as long as the list $\mathcal{P}^{\text{toStaff}}$ is not empty.

Algorithm 6.9 Sketch of the heuristic ModGRAP

```
1: Input: Instance data, output of Algorithm 6.6
2: Output: A feasible solution (x, y) for the workforce assignment problem or no solution
3: ---------------------------------------------------------------------------
4: while P^toStaff ≠ ∅ do
5:     Randomly select a project p from the list P^toStaff;
6:     if K_p^suit,rem = ∅ then
7:         terminate;                    // ModGRAP failed to construct a feasible solution
8:     Randomly select a worker k from the set K_p^suit,rem;
9:     Randomly select a skill s from the set S_kp^match,rem;
10:    Randomly select a period t ∈ T_p | r_pst^rem > 0;
11:    Allocate as much of the remaining requirement r_pst^rem as possible to worker k;
12:    Make all necessary updates of variables and sets;
```

Theorem 6.1 *The heuristic ModGRAP can find an optimal solution to any instance of the workforce assignment problem if there is an optimal solution.* □

Proof ModGRAP can construct only a subset of all feasible solutions for an instance of the workforce assignment problem. In our proof, we will show that all those solutions that cannot be constructed by ModGRAP can be transformed into solutions that can be constructed by ModGRAP and that this transformation does not deteriorate the objective function value. For a clearer presentation of our proof, we assume without loss of generality that all departmental requirements are zero.

First, realize that ModGRAP can construct only a proper subset of all feasible solutions in general. When a solution for an instance of the workforce assignment problem is determined by ModGRAP, each variable y_{kpst} is set to $\min\left(\frac{r_{pst}^{\text{rem}}}{l_{ks}}, R_{kt}^{\text{rem}}\right)$ in the course of ModGRAP, i.e., whenever workload is allocated to a worker k, as much workload as possible is allocated to k. In fact, ModGRAP can construct any feasible solution where as much workload as possible is allocated to a worker, because ModGRAP randomly selects the quadruplet (p, k, s, t) for each allocation of workload from a remaining requirement r_{pst}^{rem} to a worker k. Though, since y_{kpst} is never set to a value less than $\min\left(\frac{r_{pst}^{\text{rem}}}{l_{ks}}, R_{kt}^{\text{rem}}\right)$, ModGRAP cannot construct all feasible solutions, but only a subset of all feasible solutions.

To verify this characteristic of ModGRAP, assume that the following instance is given: $\mathcal{P} = \{p\}$, $\mathcal{K} = \{k_1, k_2\}$, $\mathcal{S} = \mathcal{S}_p = \mathcal{S}_{k_1} = \mathcal{S}_{k_2} = \{s\}$, $\mathcal{T} = \{t\}$. Additionally, let $r_{pst} = 10$,

$R_{k_1t} = R_{k_2t} = 10$, and $l_{k_1s} = l_{k_2s} = 1$. Then, a feasible solution with $y_{k_1pst} = 5$ and $y_{k_2pst} = 5$ is an example for a solution that cannot be constructed by ModGRAP, because in at least one allocation step less workload than possible was allocated. If ModGRAP is used to construct a solution for this instance, $y_{k_1pst} = 10$ or $y_{k_2pst} = 10$ must hold.

Based on the described characteristic of ModGRAP, two types of solutions of the workforce assignment problem can be distinguished. If the set of feasible solutions for an instance of the workforce assignment problem is non-empty, this set can be divided into two disjoint sets A and B. Let set A contain all feasible solutions that can be constructed by ModGRAP and let set B contain all feasible solutions that cannot be constructed by ModGRAP. Let us denote a solution of set A by $(\boldsymbol{x}^A, \boldsymbol{y}^A)$ and a solution of set B by $(\boldsymbol{x}^B, \boldsymbol{y}^B)$. Each solution $(\boldsymbol{x}^A, \boldsymbol{y}^A)$ can be generated by ModGRAP by selecting the quadruplets (p, k, s, t) in the right order and by setting the corresponding variables $y^A_{kpst} := \min\left(\frac{r^{\text{rem}}_{pst}}{l_{ks}}, R^{\text{rem}}_{kt}\right)$. In contrast, each solution $(\boldsymbol{x}^B, \boldsymbol{y}^B)$ can only be generated by a variant of ModGRAP that sets at least one variable $y^B_{kpst} < \min\left(\frac{r^{\text{rem}}_{pst}}{l_{ks}}, R^{\text{rem}}_{kt}\right)$.

In the following, we outline a five-step approach that transforms a solution $(\boldsymbol{x}^B, \boldsymbol{y}^B)$ into a solution $(\boldsymbol{x}^A, \boldsymbol{y}^A)$ whose objective function value is not worse than that of $(\boldsymbol{x}^B, \boldsymbol{y}^B)$. We consider in our proof only one period t. The five-step approach must be executed for each period $t \in \mathcal{T}$ to construct a solution $(\boldsymbol{x}^A, \boldsymbol{y}^A)$. Within the five-step approach, we gradually exclude requirements r_{pst} and variables y^B_{kpst} from consideration, while we build up solution $(\boldsymbol{x}^A, \boldsymbol{y}^A)$ simultaneously. To track the exclusion of variables y^B_{kpst}, we maintain a *reduced solution* $(\boldsymbol{x}^{B,\text{red}}, \boldsymbol{y}^{B,\text{red}})$. At the beginning of our transformation, we set $(\boldsymbol{x}^{B,\text{red}}, \boldsymbol{y}^{B,\text{red}}) := (\boldsymbol{x}^B, \boldsymbol{y}^B)$ and $\boldsymbol{x}^A := \boldsymbol{0}$; an initialization of $\boldsymbol{y}^A$ is not necessary. Whenever a variable $y^{B,\text{red}}_{kpst}$ is excluded from consideration, this variable is removed from $\boldsymbol{y}^{B,\text{red}}$. When all variables $y^{B,\text{red}}_{kpst}$ that are associated with worker k in period t are excluded from consideration, we say that worker k is excluded from consideration.

Before the start of the five-step approach, we initialize two types of variables that track two important quantities. The first type of variables tracks remaining project requirements that have not been covered by variables $\boldsymbol{y}^A$ during the five-step approach. These variables are denoted by r^{rem}_{pst}. We set $r^{\text{rem}}_{pst} := r_{pst}$ for all requirements of all projects before the five-step procedure begins. The second type of variables tracks remaining availabilities of workers during the five-step approach. These tracking variables are denoted by R^{rem}_{kt} and are initialized by setting $R^{\text{rem}}_{kt} := R_{kt}$ for all workers. After these initializations, the five-step approach can be started.

Step 1: For each variable $y^{B,\text{red}}_{kpst}$ for which $y^{B,\text{red}}_{kpst} = 0$ holds, set $y^A_{kpst} := 0$, and exclude variable $y^{B,\text{red}}_{kpst}$ from further consideration.

Step 2: For each variable $y^{B,\text{red}}_{kpst}$ for which $y^{B,\text{red}}_{kpst} = \frac{r^{\text{rem}}_{pst}}{l_{ks}}$ holds, set $y^A_{kpst} := y^{B,\text{red}}_{kpst}$, $R^{\text{rem}}_{kt} := R^{\text{rem}}_{kt} - y^{B,\text{red}}_{kpst}$, $r^{\text{rem}}_{pst} := 0$, update $\boldsymbol{x}^A$, and exclude variable $y^{B,\text{red}}_{kpst}$ and requirement r^{rem}_{pst} from further consideration.

Here, the whole requirement r^{rem}_{pst} was allocated to worker k to obtain $(\boldsymbol{x}^{B,\text{red}}, \boldsymbol{y}^{B,\text{red}})$. Hence, these allocations can also be made by ModGRAP.

Step 3: For each variable $y^{B,\text{red}}_{kpst}$ for which $y^{B,\text{red}}_{kpst} = R^{\text{rem}}_{kt}$ holds, set $y^A_{kpst} := y^{B,\text{red}}_{kpst}$, $R^{\text{rem}}_{kt} := 0$, $r^{\text{rem}}_{pst} := r^{\text{rem}}_{pst} - l_{ks} y^{B,\text{red}}_{kpst}$, update $\boldsymbol{x}^A$, and exclude variable $y^{B,\text{red}}_{kpst}$ from further con-

sideration. Since $y^{B,\mathrm{red}}_{kpst}$ was the last positive variable of worker k in the solution $\boldsymbol{y}^{B,\mathrm{red}}$, worker k is excluded from further consideration.

Here, as much of the remaining requirement r^{rem}_{pst} as possible was allocated to worker k to obtain $(\boldsymbol{x}^{B,\mathrm{red}}, \boldsymbol{y}^{B,\mathrm{red}})$. Hence, these allocations can also be made by ModGRAP.

Exclude all workers from consideration for which all variables $y^{B,\mathrm{red}}_{kpst}$ have been excluded from consideration.

Step 4: Repeat Steps 1 to 3 until a repetition occurs where no variable $y^{B,\mathrm{red}}_{kpst}$ can be excluded in any of the three steps.

If all variables $y^{B,\mathrm{red}}_{kpst}$ have been excluded from consideration: Stop, the transformation is completed.

Otherwise, there are variables $y^{B,\mathrm{red}}_{kpst}$ that have not been excluded so far. For these variables, $y^{B,\mathrm{red}}_{kpst} < \min\left(\frac{r^{\mathrm{rem}}_{pst}}{l_{ks}}, R^{\mathrm{rem}}_{kt}\right)$ holds. Each remaining requirement r^{rem}_{pst} that has not been excluded so far is accomplished by at least two workers in solution $(\boldsymbol{x}^{B,\mathrm{red}}, \boldsymbol{y}^{B,\mathrm{red}})$.

If each worker that is still considered accomplishes workload of at least two positive remaining requirements in solution $(\boldsymbol{x}^{B,\mathrm{red}}, \boldsymbol{y}^{B,\mathrm{red}})$, go to Step 5.

Else, select a worker k who contributes only to one remaining requirement r^{rem}_{pst} and an arbitrary worker k' who also contributes to the remaining requirement r^{rem}_{pst}. Then, reallocate workload of the requirement r^{rem}_{pst} from worker k' to worker k by increasing $y^{B,\mathrm{red}}_{kpst}$ by δ and decreasing $y^{B,\mathrm{red}}_{k'pst}$ by $\frac{l_{ks}}{l_{k's}}\delta$. Choose δ such that $y^{B,\mathrm{red}}_{kpst}$ becomes as large as possible, i.e., set

$$\delta := \min\left(R^{\mathrm{rem}}_{kt} - y^{B,\mathrm{red}}_{kpst},\ \frac{l_{k's}}{l_{ks}} y^{B,\mathrm{red}}_{k'pst}\right).$$

Set $y^{B,\mathrm{red}}_{kpst} := y^{B,\mathrm{red}}_{kpst} + \delta$, $R^{\mathrm{rem}}_{kt} := R^{\mathrm{rem}}_{kt} - \delta$, $y^{B,\mathrm{red}}_{k'pst} := y^{B,\mathrm{red}}_{k'pst} - \frac{l_{ks}}{l_{k's}}\delta$, and $R^{\mathrm{rem}}_{k't} := R^{\mathrm{rem}}_{k't} + \frac{l_{ks}}{l_{k's}}\delta$.

Now, either the equation $y^{B,\mathrm{red}}_{kpst} = R^{\mathrm{rem}}_{kt}$ or the equation $y^{B,\mathrm{red}}_{k'pst} = 0$ holds or both equations hold. If $y^{B,\mathrm{red}}_{k'pst} = 0$ holds, go to Step 1. Otherwise, go to Step 3.

Step 5: At this point, each remaining requirement r^{rem}_{pst} that has not been excluded so far is accomplished by at least two workers in solution $(\boldsymbol{x}^{B,\mathrm{red}}, \boldsymbol{y}^{B,\mathrm{red}})$, and each worker that is still considered accomplishes workload of at least two positive remaining requirements in solution $(\boldsymbol{x}^{B,\mathrm{red}}, \boldsymbol{y}^{B,\mathrm{red}})$. Since $\sum_{p\in\mathcal{P}^{\mathrm{suit}}_k(t)} \sum_{s\in\mathcal{S}^{\mathrm{match}}_{kp}} y^{B,\mathrm{red}}_{kpst} = R^{\mathrm{rem}}_{kt}$ can hold for each worker that is still considered, the following reallocation of workload ensures that availabilities are regarded.

We increase an already positive contribution $y^{B,\mathrm{red}}_{kpst}$ of a worker k by δ and reduce another contribution $y^{B,\mathrm{red}}_{kp's't}$ of the same worker by δ. Here, $p = p'$ or $s = s'$ is possible. Note that the variable that is increased cannot be selected arbitrarily, as we will see later on. The changes in the variables $y^{B,\mathrm{red}}_{kpst}$ and $y^{B,\mathrm{red}}_{kp's't}$ must be compensated by changes of other variables. The simplest possible way of

compensation that may exist features only one other worker k' whose contribution to project p and skill s is decreased and whose contribution to project p' and skill s' is increased. In this simplest case, only two workers are involved in reallocating workload and only two skill requirements are affected by the reallocation. Though, it may be necessary that three or more workers are involved in the reallocation. In general, the reallocation involves n workers and affects n skill requirements, $n \geq 2$. The reallocation requires for each worker who is involved that two contributions are changed. One contribution is increased, the other is decreased.

For all workers involved in reallocation, we have to ensure that their remaining availabilities are observed. In order to observe remaining availabilities, the contribution that is increased cannot be chosen arbitrarily in general, as we will see in two examples. We will give an example involving two workers and an example involving three workers. From these examples it is easy to see how the changes in contributions must be made in cases where more than three workers are involved in reallocation.

For the first example, assume that worker k_1 and worker k_2 accomplish the requirements r^{rem}_{pst} and $r^{\text{rem}}_{p's't}$. Without loss of generality, we assume that no other workers contribute to these requirements. If $l_{k_1 s} \geq \frac{l_{k_1 s'}}{l_{k_2 s'}} l_{k_2 s}$ holds, we select variable $y^{B,\text{red}}_{k_1 pst}$ in order to increase this variable. Otherwise, we increase variable $y^{B,\text{red}}_{k_1 p's't}$. To show why this selection of the variable that is increased works, we consider the case where $y^{B,\text{red}}_{k_1 pst}$ is increased. The underlying situation and the corresponding changes of the variables are illustrated in Figure 6.1. In this figure, the white boxes represent the variables $y^{B,\text{red}}_{k_1 pst}$ and $y^{B,\text{red}}_{k_1 p's't}$, which are the contributions of k_1, and the black boxes represent the variables $y^{B,\text{red}}_{k_2 pst}$ and $y^{B,\text{red}}_{k_2 p's't}$, which are the contributions of k_2.

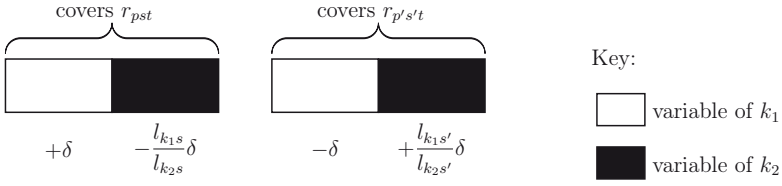


Figure 6.1: Changes in variables in Step 5 if two workers k_1 and k_2 are involved in the reallocation of workload

The increases of contributions, which are indicated by "+" in Figure 6.1, must be compensated by decreases, which are indicated by "−". For each worker who is involved, the net change in contributions must be less than or equal to 0 in order to observe the worker's availability. For worker k_1, the net change is equal to $\delta - \delta$ and thus zero. For worker k_2, the restriction $-\frac{l_{k_1 s}}{l_{k_2 s}}\delta + \frac{l_{k_1 s'}}{l_{k_2 s'}}\delta \leq 0$ must

hold, i.e., the restriction

$$l_{k_1 s} \geq \frac{l_{k_1 s'}}{l_{k_2 s'}} l_{k_2 s} \qquad \Leftrightarrow \qquad \frac{l_{k_1 s}}{l_{k_2 s}} \geq \frac{l_{k_1 s'}}{l_{k_2 s'}} \tag{6.8}$$

must be satisfied.

If Restriction (6.8) holds, i.e., if worker k_1 has a comparative advantage in exercising skill s, or, strictly speaking, no comparative disadvantage, we increase variable $y^{B,\text{red}}_{k_1 pst}$.[1] If Restriction (6.8) does not hold, worker k_1 has a comparative advantage in skill s' and we increase variable $y^{B,\text{red}}_{k_1 p' s' t}$. Since in this case $l_{k_2 s} \geq \frac{l_{k_2 s'}}{l_{k_1 s'}} l_{k_1 s}$ automatically holds (even with $>$), availabilities are observed.

The amount δ of the increase in the selected contribution is set to the maximum value that guarantees that no contribution becomes negative. Hence, we set δ to the minimum value where at least one contribution that is decreased becomes zero.

For the second example, let three workers k_1, k_2, and k_3 accomplish three requirements r^{rem}_{pst}, $r^{\text{rem}}_{p's't}$, $r^{\text{rem}}_{\bar{p}\bar{s}t}$, as depicted in Figure 6.2. Assume that variable $y^{B,\text{red}}_{k_1 pst}$ is selected in order to increase this variable. The resulting changes in other variables are given in Figure 6.2, which illustrates the reallocation of workload.

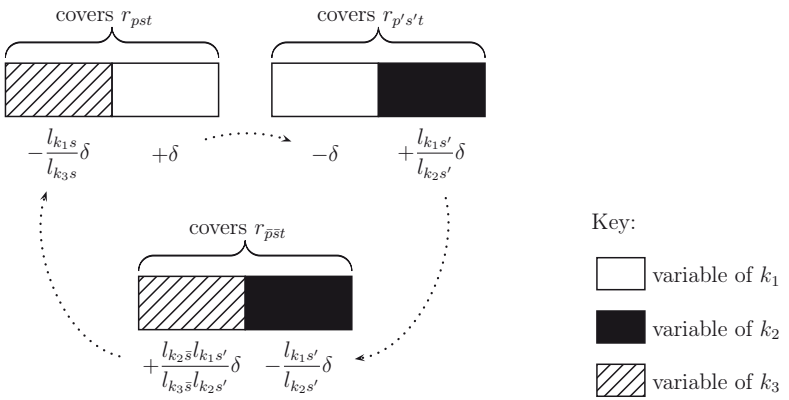


Figure 6.2: Changes in variables in Step 5 if three workers k_1, k_2, and k_3 are involved in the reallocation of workload

Again, the net change in time that a worker contributes to the three requirements must not be positive for each of the three workers who are involved, because

[1] The term "comparative advantage" was coined in the field in economics for the case of bilateral trade with two commodities, see e.g., Krugman et al. (2012, pp. 54–79), Behrens and Kirspel (2003, pp. 48–52), and Bofinger (2011, pp. 31–44).

otherwise a worker's availability could be exceeded. For workers k_1 and k_2, the net change is obviously zero. For worker k_3 the net change is not positive if

$$l_{k_1 s} \geq \frac{l_{k_2 \bar{s}} l_{k_1 s'}}{l_{k_3 \bar{s}} l_{k_2 s'}} l_{k_3 s} \tag{6.9}$$

holds. If Restriction (6.9) does not hold, we decrease variable $y^{B,\mathrm{red}}_{k_1 pst}$ by δ and increase variable $y^{B,\mathrm{red}}_{k_1 p' s' t}$ by δ, because the resulting changes do not lead to a positive net change for any of the three workers who are involved in the reallocation of workload.[2] Again, the value of δ is set to the minimum value where at least one contribution becomes zero.

As can be concluded from our two examples, Step 5 is executed in the general case as follows. Select a variable $y^{B,\mathrm{red}}_{kpst}$ of a worker k. Assume that $y^{B,\mathrm{red}}_{kpst}$ is increased by $\delta > 0$. Select another variable $y^{B,\mathrm{red}}_{kp's't}$ of the same worker k and assume that it is decreased by δ. Propagate the resulting changes in other variables until a cycle is closed. A cycle is closed when every increase (decrease) in a variable is compensated by a decrease (increase) in another variable and when all requirements are covered. When a cycle is closed, for each worker who is involved in the reallocation, two of his variables are changed, and for each requirement that is affected by the reallocation, two related variables are changed. Let the worker that closes the cycle be denoted by $\bar{k}$. Worker $\bar{k}$ is the worker that accomplishes the requirement r^{rem}_{pst} together with worker k. Check for worker $\bar{k}$ if the net change in his two variables that are affected is not positive. If the net change is positive, reverse the direction of change for each variable of all the workers who are involved in reallocation, i.e., instead of increasing (decreasing) a variable decrease (increase) the variable by the same amount. The amount of the change in each variable depends on δ. Calculate δ such that at least one variable becomes zero, but no variable becomes negative. Change the variables accordingly and go to Step 1.

The five-step approach transforms a solution $(\boldsymbol{x}^B, \boldsymbol{y}^B)$ into a solution $(\boldsymbol{x}^A, \boldsymbol{y}^A)$ whose objective function value is not worse than that of solution $(\boldsymbol{x}^B, \boldsymbol{y}^B)$. In the five-step approach, a variable y^A_{kpst} will be set to a positive value if and only if the corresponding variable $y^{B,\mathrm{red}}_{kpst}$ is positive at the time when its value is assigned to y^A_{kpst}. In addition, especially in Steps 4 and 5, only variables $y^{B,\mathrm{red}}_{kpst}$ whose corresponding variables y^B_{kpst} are positive are increased. Thus, the objective function value of the solution $(\boldsymbol{x}^A, \boldsymbol{y}^A)$, which is obtained by transformation from the solution $(\boldsymbol{x}^B, \boldsymbol{y}^B)$, cannot be worse than the objective function value of $(\boldsymbol{x}^B, \boldsymbol{y}^B)$.

Any solution from the set B can be transformed to a corresponding solution from the set A by the five-step approach as just described. Since every solution from the set A

[2] As in the first example, we have to determine whether worker k_1 should concentrate on skill s or on skill s'. It is guaranteed that one of these two alternatives is feasible with respect to worker availabilities. Restriction (6.9) represents the condition for a "comparative advantage" of worker k_1 in skill s over skill s' for the given assignment of workers to skill requirements in our three-worker, three-requirement case. Restriction (6.9) is analog to the condition for a "comparative advantage" in the multi-country, multi-commodity case in international trade (cf. Jones, 1961, especially Section 5).

can be constructed by ModGRAP, it is guaranteed that ModGRAP can find an existing optimal solution. ■

Although ModGRAP can find an existing optimal solution for any instance, we expect that its average solution quality is far worse than the average solution quality of GRAP, because the probability of scattering workers across projects is relatively high for ModGRAP in comparison with GRAP. The performance of ModGRAP will be evaluated in Subsection 7.3.2.

6.2.2 An iterated simultaneous assignment procedure (ISAP)

In this subsection, we will outline a second construction heuristic for the workforce assignment problem. We call this heuristic *iterated simultaneous assignment procedure* (ISAP). In ISAP, we calculate for each potential assignment (k, p) of a worker k to a project p a (dynamic) value $suit_{kp}$ that represents the suitability of k for p or the fit between k and p. The basic idea of ISAP is to simultaneously select one worker for each project with unallocated workload such that no worker is selected for more than one project and such that the overall fit of the resulting pairs of projects and workers is maximized. Then, for each pair (k, p) as much workload of project p as possible is allocated to the selected worker k. The procedure of calculating the values $suit_{kp}$, selecting workers and allocating workload is iterated until the requirements of all projects are covered or until a partial solution cannot be extended to a feasible solution.

In the following, we will first explain the motivation for ISAP and its relation to GRAP. Thereupon, we will describe ISAP in a formal way before we focus on the calculation of the values $suit_{kp}$ and on the problem of selecting one worker for each project with uncovered workload. Then, a multi-start version of ISAP will be outlined in brief. Finally, we will discuss some properties of ISAP.

ISAP can be thought of as an advancement of GRAP. The sequential staffing of projects in GRAP is replaced by a simultaneous, parallel staffing of all projects in ISAP. All other parts of both heuristics are essentially the same. In both heuristics, each project is staffed gradually, because workers are assigned step by step to a project. ISAP requires the same variables and sets as GRAP to track remaining requirements, remaining availabilities, and so on. The allocation of workload of a project p to a worker k and the subsequent updates in ISAP are also essentially identical with GRAP. The main difference between ISAP and GRAP are the simultaneous assignments of workers to all projects with remaining requirements in ISAP compared to the sequential staffing of projects in GRAP.

To see that simultaneous assignments to all projects can be advantageous, consider the following example.

Example 6.7 Imagine an instance of the workforce assignment problem with two projects p_1 and p_2, and two workers k_1 and k_2. For this instance, we make four assumptions:

(1) Worker k_1 can completely cover project p_1 if only workload of p_1 is allocated to him.

(2) Worker k_1 can completely cover project p_2 if only workload of p_2 is allocated to him.

(3) Worker k_2 can completely cover project p_1 if only workload of p_1 is allocated to him.

(4) Worker k_2 cannot completely cover project p_2.

Assume that the suitability values for the pairs of projects and workers are as follows: $suit_{k_1p_1} = 10$, $suit_{k_1p_2} = 7$, $suit_{k_2p_1} = 9$, $suit_{k_2p_2} = 4$. The corresponding problem of selecting at most one worker for each project such that the total suitability is maximized can be formulated as a classical assignment problem. This assignment problem is illustrated in Figure 6.3. An assignment with maximum total suitability where each project is assigned to at most one worker and each worker is assigned to at most one project is given by the pairs (k_1, p_2) and (k_2, p_1). For each of these two pairs, ISAP allocates as much workload of the project as possible to the respective worker. Hence, ISAP yields a feasible solution for the instance with an objective function value of 2.

Now, assume that we apply GRAP. Let GRAP select project p_1 as the first project to staff. For the selection of a worker, let the selection probability be proportional to the suitability values. Thus, let the selection probabilities of worker k_1 and k_2 be $\frac{10}{19}$ and $\frac{9}{19}$, respectively. Then, rather k_1 than k_2 is selected for project p_1. If GRAP selects worker k_1 and allocates as much workload of p_1 as possible to k_1, then k_1 and k_2 must be assigned to project p_2. In total, GRAP is likely to obtain a solution whose objective function value is 3. For the given instance, GRAP is likely to generate a solution that is worse than the solution generated by ISAP. □

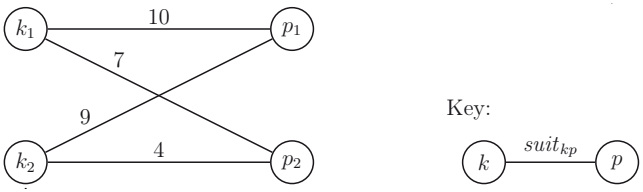


Figure 6.3: Illustration of the assignment problem encountered by ISAP in the instance of Example 6.7

Example 6.7 indicates that ISAP is less myopic than GRAP. GRAP selects a "best pair" (k, p) for a single project p, but ignores the impact of this selection on staffing possibilities of other projects. ISAP, in contrast, takes all projects into account and determines a set of best pairs.

Having explained the idea behind ISAP, we will outline this procedure in a more formal way now. ISAP can be divided into two parts: a preparatory part and a main part. The preparatory part starts just as the one of GRAP with declarations and initializations of tracking variables and sets. Hence, at the outset of ISAP, Algorithm 6.6 is invoked. Additionally, the preparatory part entails declarations and initializations that are necessary for repeatedly solving a selection problem in the main part of ISAP. In the solution of a selection problem, at most one worker is selected for each project with unallocated workload.

The main part of ISAP is summarized in Algorithm 6.10. The main part iterates the process of simultaneously selecting at most one worker for each project in $\mathcal{P}^{\text{toStaff}}$ and

allocating workload to the selected workers, as long as there are projects with uncovered requirements. At the outset of each iteration, we check if the current partial solution can be extended to a feasible solution. If there is a project $p \in \mathcal{P}^{\text{toStaff}}$ in a current partial solution for which no worker is left that can cover remaining requirements of project p, this partial solution cannot be extended to a feasible solution and Algorithm 6.10 terminates without returning a feasible solution.

Algorithm 6.10 Main part of the heuristic ISAP

1: **Input:** Instance data, output of Algorithm 6.6, further initializations
2: **Output:** A feasible solution $(\boldsymbol{x}, \boldsymbol{y})$ for the workforce assignment problem or no solution
3: ---
4: **while** $\mathcal{P}^{\text{toStaff}} \neq \emptyset$ **do**
5: **for all** $p \in \mathcal{P}^{\text{toStaff}}$ **do**
6: **if** $\mathcal{K}_p^{\text{suit,rem}} = \emptyset$ **then**
7: **terminate**; // ISAP failed to construct a feasible solution
8: **for all** $k \in \mathcal{K}_p^{\text{suit,rem}}$ **do**
9: Calculate a suitability value $suit_{kp}$; // Call Algorithm 6.8, for example
10: Formulate and solve the problem of selecting at most one worker for each project $p \in \mathcal{P}^{\text{toStaff}}$;
11: **for all** $p \in \mathcal{P}^{\text{toStaff}}$ **do**
12: **if** a worker $k \in \mathcal{K}_p^{\text{suit,rem}}$ was selected for project p **then**
13: **while** $\mathcal{S}_{kp}^{\text{match,rem}} \neq \emptyset$ **do**
14: Randomly select a skill s from the set $\mathcal{S}_{kp}^{\text{match,rem}}$;
15: $totSkillContr := 0$;
16: Allocate as much of the remaining requirements of project p for skill s as possible to worker k;
17: Record the time that worker k contributes to skill s in the variable $totSkillContr$;
18: **if** $totSkillContr > 0$ **then**
19: $x_{kp} := 1$;
20: Make all necessary updates of variables and sets;

If there is at least one remaining suitable worker for each project $p \in \mathcal{P}^{\text{toStaff}}$, we calculate a non-negative suitability value $suit_{kp}$ for all remaining potential assignments (k, p), $p \in \mathcal{P}^{\text{toStaff}}$, $k \in \mathcal{K}_p^{\text{suit,rem}}$ (cf. line 9 of Algorithm 6.10). The same suitability values as in GRAP can be applied, for example. We will elaborate on the suitability values after our overview of Algorithm 6.10. Right now, we proceed with the overview.

Given the suitability values, we formulate the problem of selecting at most one worker for each project p with remaining requirements such that no worker is selected for more than one project (cf. line 10). The objective of the problem is to find a selection that has maximum total suitability. The solution to this selection problem can exhibit projects for which no remaining suitable worker was selected. This can happen if and only if there is a subset $\bar{\mathcal{P}}^{\text{toStaff}} \subset \mathcal{P}^{\text{toStaff}}$ for which $\left|\bigcup_{p \in \bar{\mathcal{P}}^{\text{toStaff}}} \mathcal{K}_p^{\text{suit,rem}}\right| < \left|\bar{\mathcal{P}}^{\text{toStaff}}\right|$ holds. To give an example for this case, assume that there are two projects p_1 and p_2 and that worker k is

the last remaining suitable worker for both projects. Then, worker k is selected for either p_1 or p_2.

For each project $p \in \mathcal{P}^{\text{toStaff}}$ for which a worker $k \in \mathcal{K}_p^{\text{suit,rem}}$ was selected, we allocate as much workload of project p as possible to worker k. This workload allocation is done in the same way as in GRAP. We repeatedly select one remaining matching skill between k and p (cf. line 14). Like in GRAP, we choose one of the best remaining matching skills, say skill s. For this skill s, we allot for each period $t \in \mathcal{T}_p$ as much workload as possible to worker k (cf. lines 13–19 of Algorithm 6.7 on page 131). If in any period a positive amount of workload can be allocated to worker k, the allocation is carried out and we set $x_{kp} := 1$. Then, we update all other variables and sets as in GRAP (cf. lines 22–34 of Algorithm 6.7). After the updates, Algorithm 6.10 continues either with selecting another remaining matching skill, or with allocating workload of the next project for that a worker was selected, or with formulating the new selection problem if there are projects with unallocated workload. We will evaluate the time complexity of Algorithm 6.10 when we have elaborated on the formulation and solution of the selection problem.

The suitability values $suit_{kp}$, $p \in \mathcal{P}^{\text{toStaff}}$, $k \in \mathcal{K}_p^{\text{suit,rem}}$, must be calculated for each selection problem that is formulated in the course of ISAP (cf. line 9 of Algorithm 6.10). The purpose of the values $suit_{kp}$ is to indicate how helpful an assignment of worker k to project p would be in order to yield a solution with as few assignments in total as possible. Since we do not know in advance which assignments are helpful and which are not, we can only estimate how helpful or suitable an assignment is at a certain point during the ISAP procedure. Manifold possibilities exist to make these estimates, hence many possibilities exist to calculate the values $suit_{kp}$. We decided to use the same suitability values as for GRAP, i.e., we use either $suit_{kp}^{\text{A}}$, which is specified in Definitions (6.7), or $suit_{kp}^{\text{B}}$, which is computed by Algorithm 6.8.

Given the suitability values $suit_{kp}$, we can formulate the problem of selecting at most one remaining suitable worker for each project in the list $\mathcal{P}^{\text{toStaff}}$ (cf. line 10 of Algorithm 6.10). We provide two formulations for this selection problem. The first formulation is an intuitive formulation that can be solved by the simplex method. We used a general-purpose LP solver from the solver package CPLEX that applies the dual simplex method. The second formulation models the selection problem as a classical assignment problem. The classical assignment problem can be solved by the standard simplex method, but also by more efficient methods, which exploit its special structure. For both formulations, we need non-negative continuous decision variables $a_{kp} \in [0, 1]$ that indicate whether worker k is selected for project p ($a_{kp} = 1$) or not ($a_{kp} = 0$).

The first formulation of the selection problem is given by (6.10)–(6.13).

$$\text{Max.} \quad \sum_{p \in \mathcal{P}^{\text{toStaff}}} \sum_{k \in \mathcal{K}_p^{\text{suit,rem}}} (4SP + suit_{kp})\, a_{kp} \tag{6.10}$$

$$\text{s. t.} \quad \sum_{k \in \mathcal{K}_p^{\text{suit,rem}}} a_{kp} \leq 1 \qquad p \in \mathcal{P}^{\text{toStaff}} \tag{6.11}$$

$$\sum_{p \in \mathcal{P}^{\text{toStaff}} \mid k \in \mathcal{K}_p^{\text{suit,rem}}} a_{kp} \leq 1 \qquad k \in \bigcup_{p \in \mathcal{P}^{\text{toStaff}}} \mathcal{K}_p^{\text{suit,rem}} \tag{6.12}$$

$$a_{kp} \in [0, 1] \qquad p \in \mathcal{P}^{\text{toStaff}},\ k \in \mathcal{K}_p^{\text{suit,rem}} \tag{6.13}$$

Objective function (6.10) maximizes the total suitability of all selected pairs (k, p). The constant term $4SP$ that we add to each suitability value ensures that the maximum number of assignments is realized, i.e., that the maximum number of pairs is selected.[3] Without such a term, it might be better to select one very suitable pair (k, p) instead of two pairs (k, p') and (k', p) whose total suitability is less than that of the pair (k, p). Constraints (6.11) ensure that for each project $p \in \mathcal{P}^{\text{toStaff}}$ at most one worker is selected who belongs to the set of remaining suitable workers for project p. Constraints (6.12) guarantee that each worker k who can cover unallocated workload is selected for at most one project to which he can contribute. The domain of the decision variables is stated in Constraint set (6.13).

Note that we need not demand the decision variables a_{kp} to be binary variables, because special properties of model (6.10)–(6.13) ensure that all solutions that are considered by the simplex method, i.e., all basis solutions, are integer. The special properties that are responsible for integer basis solutions lie in the Constraint sets (6.11) and (6.12), which can be expressed as $\boldsymbol{Ax} \leq \boldsymbol{b}$. Since for (6.11) and (6.12) matrix $\boldsymbol{A}$ is totally unimodular, the right-hand side $\boldsymbol{b}$ is integer, and the decision variables are bounded by integers, all basis solutions of model (6.10)–(6.13) are integer (cf. Burkard et al., 2009, pp. 29–30 and 74–75; Neumann and Morlock, 2002, pp. 310 and 384–386).

The second formulation of the selection problem represents the problem as a classical assignment problem in order to exploit efficient solution methods that were developed for the classical assignment problem. Burkard et al. (2009, pp. 73–144) give an overview of solution methods for the classical assignment problem, which they call linear sum assignment problem. A formulation of our selection problem within ISAP as a classical assignment problem requires that we select exactly one worker for each project and that each worker is selected exactly once. Hence, we must assure that the number of projects is equal to the number of workers. To fulfill this prerequisite, we introduce the sets $\bar{\mathcal{P}}$ and $\bar{\mathcal{K}}$ that contain all projects $p \in \mathcal{P}$ and all workers $k \in \mathcal{K}$, respectively. If $|\mathcal{P}| < |\mathcal{K}|$ ($|\mathcal{K}| < |\mathcal{P}|$) the set $\bar{\mathcal{P}}$ ($\bar{\mathcal{K}}$) is stocked up by dummy projects (dummy workers) such that $|\bar{\mathcal{P}}| = |\bar{\mathcal{K}}|$ holds.

Additionally, we assure that each project $p \in \bar{\mathcal{P}}$ can be assigned to each worker $k \in \bar{\mathcal{K}}$. Consequently, we must define a suitability value $suit_{kp}$ for each pair (k, p), $p \in \bar{\mathcal{P}}$, $k \in \bar{\mathcal{K}}$. For all pairs (k, p), $p \in \mathcal{P}^{\text{toStaff}}$, $k \in \mathcal{K}_p^{\text{suit,rem}}$, the suitability values are calculated as in the first formulation of the selection problem. For all other pairs (k, p), we set $suit_{kp} := -4SP$. This large negative value ensures that as many projects as possible are assigned to real workers. Without the large negative value, it might be advantageous to make the assignments (k, p) and (k', p') where worker k' is a dummy worker for project p', although the assignments (k, p') and (k', p) would represent pairs of real workers and real projects.

The formulation of the selection problem as a linear sum assignment problem is given by (6.14)–(6.17).

[3] The constant term $4SP$ takes into account the maximum possible difference in suitability values, which is bounded by $2S$. This bound applies to $suit_{kp}^{\text{A}}$, which exhibits the maximum possible difference among our suitability values. The additional factor 2 considers the perturbation of suitability values that is deployed for the multi-start approach (cf. Equation 6.18 on page 147). Eventually, the factor P regards the extreme case where $P - 1$ pairs, each with maximum suitability value, would be selected instead of P pairs, each with minimum suitability value.

$$\text{Max.} \quad \sum_{p\in\bar{\mathcal{P}}}\sum_{k\in\bar{\mathcal{K}}} suit_{kp}a_{kp} \tag{6.14}$$

$$\text{s. t.} \quad \sum_{k\in\bar{\mathcal{K}}} a_{kp} = 1 \qquad p \in \bar{\mathcal{P}} \tag{6.15}$$

$$\sum_{p\in\bar{\mathcal{P}}} a_{kp} = 1 \qquad k \in \bar{\mathcal{K}} \tag{6.16}$$

$$a_{kp} \in [0,1] \qquad p \in \bar{\mathcal{P}},\ k \in \bar{\mathcal{K}} \tag{6.17}$$

Objective function (6.14) maximizes the total suitability of all assignments (k, p). Constraints (6.15) ensure that each real project and each dummy project (if there is any) is assigned to exactly one worker, be it a real or a dummy worker. Constraints (6.16) guarantee that each real worker and each dummy worker (if there is any) is assigned to exactly one project, be it a real or a dummy project. The domain of the decision variables is stated in Constraint set (6.17). Again, we need not demand $a_{kp} \in \{0, 1\}$, because the same special properties hold for (6.14)–(6.17) as for (6.10)–(6.13).

The linear sum assignment problem (6.14)–(6.17) can also be represented by a bipartite graph whose edges connect nodes that represent projects with nodes that represent workers. The weight of an edge is given by the corresponding suitability value. Then, an optimal solution of our linear sum assignment problem corresponds to a perfect matching of maximum suitability in this bipartite graph.

Many methods have been proposed for solving the linear sum assignment problem. The standard simplex method and the network simplex method suffer from a high number of degenerate pivot operations when applied to the assignment problem (cf. Barr et al., 1977; Engquist, 1982, p. 371; Burkard et al., 2009, pp. 31, 104, and 106–112). An efficient class of methods are so called successive shortest path methods (Burkard et al., 2009, pp. 93–104). We implemented the successive shortest path method that has been outlined by Glover et al. (1986, pp. 12–19) in the first part of their paper. For a neat presentation of this method, see Neumann and Morlock (2002, pp. 294–300).

To apply the successive shortest path algorithm of Glover et al. (1986, pp. 12–19), the assignment problem under consideration is represented by a bipartite directed graph $G = (N, A)$ with node set N and arc set A. In our case, we establish one node for each project $p \in \bar{\mathcal{P}}$. We call these nodes *project nodes*. Additionally, we establish one node for each worker $k \in \bar{\mathcal{K}}$. These nodes are called *worker nodes*. Hence, the node set N is given by $\bar{\mathcal{P}} \cup \bar{\mathcal{K}}$. From each project node, arcs run to all worker nodes, thus, the arc set A contains $|\bar{\mathcal{P}}||\bar{\mathcal{K}}|$ arcs. The length of an arc $\langle p, k\rangle$ from project node p to worker node k is set to $suit_{kp}$. At the outset, no assignments between projects and workers exist. Put in other words, no assignments between project nodes and worker nodes exist.

Then, at most $|\bar{\mathcal{P}}|$ iterations are executed in each of which the number of assignments increases by at least one. However, assignments—except for those made in the final iteration—are provisional because in an iteration reassignments are possible. In each iteration, we apply a label correcting procedure to determine shortest paths to all unassigned worker nodes among those paths that start from unassigned project nodes. Based on shortest paths information, at least one additional assignment and potential reassignments are made. Before the next iteration starts, some arcs are reversed and

their length value is updated. The iterations stop when each project node is assigned to a worker node. At this point an optimal solution of the assignment problem has been constructed. For details of the successive shortest path method, we refer to Glover et al. (1986, pp. 12–19) and Neumann and Morlock (2002, pp. 294–300). The running time of our successive shortest path algorithm is $O\left((|\bar{\mathcal{P}}| + |\bar{\mathcal{K}}|)^4\right) = O\left((2 \cdot \max(P, K))^4\right)$ (cf. Neumann and Morlock, 2002, p. 300).

Now, as all parts of Algorithm 6.10 have been outlined, we can evaluate the time complexity of Algorithm 6.10. We formulated the selection problem in Algorithm 6.10 as a linear sum assignment problem and applied the outlined successive shortest path algorithm to solve this problem. With these specifications and assuming—as for the time complexity of GRAP—that suitability values are randomly determined and that matching skills are randomly selected, Algorithm 6.10 runs in $O\left(K\left(PSTK + (2 \cdot \max(P, K))^4\right)\right)$ time.

For a multi-start version, we directly embedded ISAP in the generic multi-start procedure outlined in Algorithm 6.5 on page 126. This multi-start version of ISAP constructs different solutions, because remaining matching skills are randomly selected for allocation of workload within ISAP. Though, these solutions are quite similar to one another; they are not scattered across the solution space. A diversification strategy helps to explore regions of the solution space that may be ignored otherwise. In case of ISAP, perturbing the suitability values is an example for a diversification strategy. Perturbing suitability values leads to a stronger randomization of ISAP. We implemented this perturbation strategy as follows. Immediately after having calculated a suitability value $suit_{kp}$ (cf. line 9 of Algorithm 6.10), we set

$$suit_{kp} := [0.5 + 1.5 \cdot \text{rand}(0, 1)] \cdot suit_{kp} \tag{6.18}$$

where $\text{rand}(0, 1)$ represents a random number that is uniformly distributed in the interval $[0, 1)$. Note that this perturbation is only applied to suitability values $suit_{kp}$ where $p \in \mathcal{P}^{\text{toStaff}}$ and $k \in \mathcal{K}_p^{\text{suit,rem}}$.

Finally, an important property of ISAP shall be mentioned. Like GRAP, the heuristic ISAP can fail to construct any existing feasible solution for instances of the workforce assignment problem, even if arbitrary suitability values are allowed during the whole course of ISAP. The instance that we described in the proof of Proposition 6.1 on page 134 is an example for an instance for which ISAP cannot find an existing feasible solution. Unlike GRAP, there is no modified version of ISAP that overcomes this drawback without sacrificing the key feature of ISAP. This key feature are simultaneous assignments. Assume a modified version of ISAP that still features simultaneous assignments, but that allocates only the workload of one skill and one period from each project to the selected worker, as it is done in ModGRAP. In the following example, we present an instance for which the unique feasible solution cannot be found by such a modified version of ISAP.

Example 6.8 Consider an instance with three projects p_1, p_2, p_3, three workers k_1, k_2, k_3, and three skills s_1, s_2, s_3. All projects last only for one period and must be executed in period t. Let projects p_1 and p_2 require only skill s_1 with $r_{p_1 s_1 t} = r_{p_2 s_1 t} = 5$, while project p_3 requires only skills s_2 and s_3 with $r_{p_3 s_2 t} = r_{p_3 s_3 t} = 10$. Let $\mathcal{S}_{k_1} = \{s_1\}$ with $l_{k_1 s_1} = 1$, $\mathcal{S}_{k_2} = \{s_1, s_2\}$ with $l_{k_2 s_1} = l_{k_2 s_2} = 1$, and $\mathcal{S}_{k_3} = \{s_1, s_3\}$ with $l_{k_3 s_1} = l_{k_3 s_3} = 1$. For the availabilities of the workers, assume $R_{k_1 t} = R_{k_2 t} = R_{k_3 t} = 10$.

The unique feasible solution of this instance requires that worker k_1 accomplishes the complete workload of projects p_1 and p_2, while workers k_2 and k_3 accomplish the whole workload of project p_3.

If the modified version of ISAP that we outlined above is applied to this instance, the unique feasible solution cannot be found. In the first iteration, the modified version of ISAP would assign either worker k_2 or worker k_3 to a project out of the set $\{p_1, p_2\}$ and allocate workload. Since in the unique feasible solution only worker k_1 is assigned to projects p_1 and p_2, the assignments made by the modified version of ISAP in the first iteration cannot lead to this solution. □

The two heuristics that we present next overcome the drawback of GRAP and ISAP that existing feasible solutions can be missed. If the set of feasible solutions for an instance is non-empty, the following two heuristics always return a feasible solution.

6.2.3 A drop procedure (DROP)

In this subsection, we will present a *drop procedure* to heuristically solve the workforce assignment problem. We call the procedure DROP, because this procedure applies the principle of drop methods. The basic idea of DROP is to start with a feasible solution in which each worker is assigned to every project for which he is suitable. Then, assignments (k, p) are repeatedly dropped, i.e., canceled, until dropping any further assignment would lead to infeasibility. To check whether dropping an assignment is feasible or not, it is sufficient to solve an LP with constant objective function value. This LP results from the fact that all binary variables x_{kp} are fixed to either 0 or 1 in the course of DROP.[4]

In Subsection 6.2.3.1, we will address the origin of drop methods and discuss related methods before we give an overview of our heuristic DROP. Subsequently, we will elaborate on the probability of selecting an assignment (k, p) for dropping and on two extensions of DROP that can considerably accelerate this heuristic. Finally, we succinctly describe a multi-start version of DROP. In Subsections 6.2.3.2 and 6.2.3.3, another way of accelerating DROP is presented. This acceleration comprises two intertwining steps. In the first step, we formulate the linear programs that must be solved within DROP as generalized network flow problems. The second step concerns solution methods for this type of problems. For generalized network flow problems, the generalized network simplex method is a very efficient solution method. We describe how to formulate generalized network flow problems for our case in Subsection 6.2.3.2. In Subsection 6.2.3.3, we will outline the generalized network simplex method and its integration into DROP. For the generalized network simplex method, we will provide detailed pseudo code for its pivot operations, which are based on manipulating network structures. To the best of our knowledge, such detailed pseudo code has not been published before. Some aspects of our implementation of the generalized network simplex method are critically discussed in Subscection 6.2.3.4. In Subsection 6.2.3.5, we discuss some properties of DROP and present a refinement of DROP derived from one property discussed.

[4] Actually, for a single LP, all x_{kp} are parameters, not variables, because the values of all x_{kp} are fixed. Nevertheless, we will stick to the term *variable* for x_{kp}, $k \in \mathcal{K}$, $p \in \mathcal{P}_k^{\text{suit}}$, because x_{kp} can be considered as a decision variable for the heuristic DROP.

6.2.3.1 Origin and outline of DROP

In this subsection, we will briefly look at the original drop method, which was applied to a location problem. Our heuristic DROP follows its idea. Furthermore, we will address methods that are closely related to drop methods and discuss their applicability to our problem. Eventually, we will outline DROP.

Our heuristic DROP was inspired by the famous drop method that Feldman et al. (1966) outlined for a warehouse location problem. The warehouse location problem is also termed plant location or facility location problem. In the classical warehouse location problem a subset of potential warehouse locations must be selected that minimizes fixed warehouse operating costs and variable shipping costs (cf. Domschke and Drexl, 1996, pp. 41–120). The drop method of Feldman et al. (1966) starts with a solution in which all warehouses are opened, i.e., in which all locations are selected. When a warehouse is closed (dropped), savings are achieved if the resulting increase in shipping costs is overcompensated by the resulting decrease in fixed warehouse operating costs. Hence, the drop method closes one warehouse at a time until no further savings are possible.

The warehouse location problem has some similarities to our workforce assignment problem. If we restrict the workforce assignment problem to a single period t, we can interpret each potential assignment (k, p) as a warehouse that can supply only one customer, namely, project p. Working time can be shipped from a warehouse (k, p) to project p. Though, in our case shipping costs are zero and, if we consider a worker k, there are no separate capacity limits for each warehouse (k, p), but there is a joint capacity limit for the warehouses (k, p), $p \in \mathcal{P}_k^{\text{suit}}$. This joint capacity limit is given by the availability R_{kt} of worker k in the sole period t. Like the workforce assignment problem, the warehouse location problem is strongly NP-hard (cf. Cornuéjols et al., 1991, pp. 289–290).

Other methods that have been devised for the warehouse location problem can also be adopted for our problem. The drop method of Feldman et al. (1966) was developed as an alternative to the add method of Kuehn and Hamburger (1963). The add method starts with a solution in which all warehouses are closed. If warehouse capacities must be observed, only a fictitious warehouse with zero fixed cost, but prohibitive shipping costs is open at the outset. Then, one warehouse at a time is opened until no further savings can be achieved. In our case, a drop method offers—compared to an add method—the advantage that we can exploit information from a feasible solution to decide about the next assignment that will be dropped. In a preliminary test of an add method for our problem, we found that the results did not surpass the results obtained by DROP. Improvement heuristics that combine add and drop concepts were also devised for the warehouse location problem (cf. Jacobsen, 1983; Whitaker, 1985). However, a preliminary test of such an interchange heuristic for our problem was not promising. Hence, we adhered to DROP.

To give an overview of DROP, we split the heuristic into a preparatory part and a main part. For both parts, we consider the network model (4.19)–(4.29) as the underlying formulation of the workforce assignment problem, however there would be no essential difference if we based DROP on the standard model. As for the description of the other heuristics, we assume that there are no workers that have already been assigned to an (ongoing) project, i.e., we assume that Constraint set (4.25) is empty.

The preparatory part of DROP is outlined in Algorithm 6.11. At the beginning, we

set each variable x_{kp}, $k \in \mathcal{K}$, $p \in \mathcal{P}_k^{\text{suit}}$, to 1 and insert the corresponding assignment (k, p) in the list $\mathcal{C}$. The list $\mathcal{C}$ contains all pairs (k, p) for which dropping, i.e., setting x_{kp} to 0, is considered later on in the main part of DROP. For now, all variables x_{kp} are tentatively fixed to 1, i.e., we have $\boldsymbol{x} = \mathbf{1}$.

Algorithm 6.11 Preparatory part of the heuristic DROP (based on the network model (4.19)–(4.29))

1: **Input:** Instance data
2: **Output:** Data required by the main part of the heuristic DROP
3: ---
4: $\mathcal{C} := \emptyset$;
5: **for all** $k \in \mathcal{K}$ **do**
6: **for all** $p \in \mathcal{P}_k^{\text{suit}}$ **do**
7: $x_{kp} := 1$;
8: $\mathcal{C} := \mathcal{C} \cup \{(k, p)\}$;
9: **for all** $t \in \mathcal{T}$ **do**
10: Formulate and solve the remaining linear program LP_t^{rem};
11: Store the values of all variables f_{kpt}^{proj} and y_{kpst};
12: **Algorithm 6.13**; // Drop assignments without contribution
13: **for all** $p \in \mathcal{P}$ **do**
14: **Algorithm 6.14**(p); // Compute LB_p^{Drop} and remove pairs from $\mathcal{C}$ if possible
15: **for all** $s \in \mathcal{S}_p$ **do**
16: **Algorithm 6.15**(p, s); // Compute LB_{ps}^{Drop} and remove pairs from $\mathcal{C}$ if possible

Before any drop operation is executed, we determine a feasible solution for the variables y_{kpst} given $\boldsymbol{x} = \mathbf{1}$. Assuming that an feasible solution exists, there exists a feasible solution with $\boldsymbol{x} = \mathbf{1}$. To find corresponding values for the variables y_{kpst}, we solve the network model (4.19)–(4.29). Since all binary variables x_{kp} are fixed, the objective function of the network model becomes a constant function, because objective function (4.19) depends only on variables x_{kp}. Hence, it is sufficient to determine any solution that satisfies all constraints of the network model to obtain values for the variables y_{kpst}. A second consequence from fixed binary variables x_{kp} is that the MIP model (4.19)–(4.29) becomes a linear program, as all remaining variables are continuous variables.

The linear program that results from fixing all binary variables x_{kp} decomposes into T linear programs, one for each period $t \in \mathcal{T}$, because the variables x_{kp} are the only elements of the MIP that couple the periods. Let LP_t^{rem} denote the linear program that remains for period t if all variables x_{kp} are fixed to either 0 or 1. For problem LP_t^{rem}, the objective function value is also fixed, thus problem LP_t^{rem} is a feasibility problem.

We formulate and solve the remaining linear program LP_t^{rem} for each period $t \in \mathcal{T}$ using the solver package CPLEX to get values for the variables f_{kpt}^{proj} and y_{kpst} of period t. Because we presume that our set of projects $\mathcal{P}$ is a feasible portfolio, we obtain a feasible solution for each linear program given $\boldsymbol{x} = \mathbf{1}$. The values of all variables f_{kpt}^{proj} and y_{kpst} are stored, as they are needed later on (cf. line 11 of Algorithm 6.11).

At the end of the preparatory part, two extensions occur, which can considerably accelerate the heuristic. The first extension calls Algorithm 6.13, the second extension

invokes Algorithm 6.14 for all projects $p \in \mathcal{P}$ and Algorithm 6.15 for all required skills of each project p. We will explain these extensions after the overview of DROP.

The main part of DROP is outlined in Algorithm 6.12. The main part starts with a non-empty list $\mathcal{C}$ and terminates when $\mathcal{C}$ is empty. As long as $\mathcal{C}$ is not empty, we compute for each pair (k, p) in $\mathcal{C}$ an unsuitability value us_{kp}. This value us_{kp} is intended to indicate how dispensable worker k is for project p. How the unsuitability values are calculated will be explained immediately after the overview of DROP. Given the unsuitability values, we determine a selection probability of each pair (k, p) in the list $\mathcal{C}$ that is proportional to the corresponding unsuitability value us_{kp}. According to the selection probabilities, we select a pair $(k, p) \in \mathcal{C}$ by a roulette wheel selection.

Algorithm 6.12 Main part of the heuristic DROP (based on the network model (4.19)–(4.29))

Input: Instance data, output of Algorithm 6.11
Output: A feasible solution $(\boldsymbol{x}, \boldsymbol{y})$ for the workforce assignment problem

while $\mathcal{C} \neq \emptyset$ **do**
  for all $(k, p) \in \mathcal{C}$ **do**
    Calculate an unsuitability value us_{kp}; // Use Equations 6.19, for example
  Randomly select a pair $(k, p) \in \mathcal{C}$ with a probability proportional to us_{kp};
  $x_{kp} := 0$;
  DropFailed := **false**;
  for all $t \in \mathcal{T}_p$ **do**
    Update the remaining linear program $\mathrm{LP}_t^{\mathrm{rem}}$; // Require $f_{kpt}^{\mathrm{proj}} = 0$
    Solve $\mathrm{LP}_t^{\mathrm{rem}}$;
    if no feasible solution for $\mathrm{LP}_t^{\mathrm{rem}}$ was found **then**
      DropFailed := **true**;
      break; // Abort the for-loop
  if *DropFailed* = **true then**
    $x_{kp} := 1$;
    for all $t \in \mathcal{T}_p$ **do**
      Update the remaining linear program $\mathrm{LP}_t^{\mathrm{rem}}$; // Require $f_{kpt}^{\mathrm{proj}} \leq R_{kt}$
    $\mathcal{C} := \mathcal{C} \setminus \{(k, p)\}$;
  else
    for all $t \in \mathcal{T}_p$ **do**
      Store the values of all variables f_{kpt}^{proj} and y_{kpst};
    $\mathcal{C} := \mathcal{C} \setminus \{(k, p)\}$;
    Algorithm 6.13; // Drop assignments without contribution
    Algorithm 6.14(p); // Compute LB_p^{Drop} and remove pairs from $\mathcal{C}$ if possible
    for all $s \in \mathcal{S}_{kp}^{\mathrm{match}}$ **do**
      Algorithm 6.15(p, s); // Compute LB_{ps}^{Drop} and remove pairs from $\mathcal{C}$ if possible

For the selected pair (k, p), we test whether setting $x_{kp} := 0$ still yields a feasible

solution. Only periods in which project p is executed are affected by this drop operation. Hence, we update the problem $\mathrm{LP}_t^{\mathrm{rem}}$ in each period $t \in \mathcal{T}_p$ (cf. line 11 of Algorithm 6.12).

If at least one updated problem $\mathrm{LP}_t^{\mathrm{rem}}$, $t \in \mathcal{T}_p$, becomes infeasible, we undo the drop operation by resetting $x_{kp} := 1$ and adjust all problems $\mathrm{LP}_t^{\mathrm{rem}}$, $t \in \mathcal{T}_p$, accordingly. Now, x_{kp} is finally fixed to 1. Since x_{kp} is finally fixed, the pair (k, p) is removed from the list $\mathcal{C}$.

If the drop operation succeeds, i.e., if all updated problems are feasible, we will store the new values of the variables f_{kpt}^{proj} and y_{kpst}; x_{kp} is finally fixed to 0 and the pair (k, p) is removed from the list $\mathcal{C}$. In case of a successful drop operation, two extensions are processed before the next drop operation starts. The two extensions are similar to those extensions of the preparatory part and they can also considerably accelerate the removal of pairs from the list $\mathcal{C}$. The first extension comprises Algorithm 6.13, the second extension comprises the Algorithms 6.14 and 6.15. In contrast to the preparatory part, the Algorithms 6.14 and 6.15 are only processed for that project p for which the drop operation associated with worker k succeeded and for the matching skills between project p and worker k.

Eventually, when the list $\mathcal{C}$ is empty, matrix $\boldsymbol{x}$ and the stored values of matrix $\boldsymbol{y}$ provide a solution for the workforce assignment problem.

The running time of Algorithm 6.12 depends on the method that is applied for solving the linear programs $\mathrm{LP}_t^{\mathrm{rem}}$. If a polynomial-time method is applied, then Algorithm 6.12 will run in polynomial time. However, we apply the simplex method or, alternatively, the generalized network simplex method. Both methods have exponential time complexity. Thus, Algorithm 6.12 has an exponential worst-case time complexity in our case. In practice, the simplex method and the generalized network simplex method are very fast. Hence, the average-case performance of Algorithm 6.12 is expected to be acceptable.

Solving the linear programs $\mathrm{LP}_t^{\mathrm{rem}}$ for those periods t that are affected by a drop operation plays a central role in DROP, because it is a recurring, time-consuming part of DROP. This part is required to check whether the variables $\boldsymbol{y}$ can be made consistent with the given values for the variables $\boldsymbol{x}$. Within DROP, the variables $\boldsymbol{y}$ are more flexible than the variables $\boldsymbol{x}$. If a variable x_{kp} is finally fixed to 1, the corresponding variables y_{kpst} will not be fixed until the very end of DROP. Only if a variable x_{kp} is finally fixed to 0, the corresponding variables y_{kpst} will be fixed immediately to 0. The flexibility of the variables y_{kpst} in the case where x_{kp} is finally fixed to 1 gives more leeway for further drop operations, as the values of the variables y_{kpst} can be adjusted by a solver to changes in $\boldsymbol{x}$.

Having given an overview of DROP, we will have a closer look at the unsuitability values us_{kp}. The unsuitability values us_{kp} are an important element of DROP. They have a major impact on the order in which assignments are dropped, because they determine the selection probabilities of those assignments that belong to the list $\mathcal{C}$. One can think of many ways on how to calculate the unsuitability values us_{kp}. A first source of inspiration can be the suitability values $suit_{kp}$ that were devised for GRAP and ISAP. The reciprocal of such a suitability value can be a good starting point for an unsuitability value. We tapped this source and tested static and dynamic unsuitability values, for which we will give three examples in the following.

Our first example is an example for a static unsuitability value. The computation of

this unsuitability value is stated in Definitions (6.19).

$$us_{kp}^{\mathrm{A}} := \frac{1}{\sum\limits_{s \in \mathcal{S}_{kp}^{\mathrm{match}}} l_{ks}} \qquad (k,p) \in \mathcal{C} \tag{6.19}$$

The definition of us_{kp}^{A} assumes that project p can the better dispense with worker k, the smaller the sum of the levels of the matching skills between worker k and project p.

Our second example is an example for a dynamic unsuitability value. This unsuitability value is specified by Definitions (6.20).

$$us_{kp}^{\mathrm{B}} := \frac{\sum\limits_{p' \in \mathcal{P}_k^{\mathrm{suit}}} x_{kp'}}{\sum\limits_{s \in \mathcal{S}_{kp}^{\mathrm{match}}} l_{ks}} \qquad (k,p) \in \mathcal{C} \tag{6.20}$$

For us_{kp}^{B}, we additionally take into account the number of projects to which worker k is currently assigned. The definition of us_{kp}^{B} results from the view that a great number of current assignments renders worker k unattractive for project p. The underlying assumption of this view is that the greater the number of assignments of worker k, the less his potential contribution to project p.

Definitions (6.19) and (6.20) can be modified by replacing the sum of the skill levels of the matching skills by the number of matching skills. Though, this and other modifications led to solutions of worse quality.

Our third and last example for a possible calculation of an unsuitability value considers a dynamic unsuitability value that exploits information from the values of the variables y_{kpst} in the current solution. Definitions (6.21) show how this dynamic unsuitability value us_{kp}^{C} is calculated.

$$us_{kp}^{\mathrm{C}} := \frac{1}{\sum\limits_{s \in \mathcal{S}_{kp}^{\mathrm{match}}} \sum\limits_{t \in \mathcal{T}_p \,|\, r_{pst} > 0} \frac{l_{ks} y_{kpst}}{r_{pst}}} \qquad (k,p) \in \mathcal{C} \tag{6.21}$$

In Definitions (6.21), the denominator considers each positive skill requirement r_{pst} of project p to which worker k can contribute and the actual contributions of worker k to these requirements in the current solution $\boldsymbol{y}$. The ratios of contributions to requirements are summed up over all relevant periods for all matching skills between project p and worker k. The reciprocal of this sum defines the unsuitability value, which deems worker k the more unsuitable for project p, the smaller his relative contributions are in the current solution. Note that Definitions (6.21) cannot be applied in the case where $\sum_{s \in \mathcal{S}_{kp}^{\mathrm{match}}} \sum_{t \in \mathcal{T}_p} y_{kpst} = 0$ holds; though, Algorithm 6.13 guarantees that this case cannot occur, as we will see in the next paragraph.

Algorithm 6.13 is the first of our two extensions that help to accelerate DROP. We invoke Algorithm 6.13 whenever a new feasible solution of the workforce assignment problem was found. This algorithm scans the list $\mathcal{C}$ and spots pairs (k,p) where worker k does not accomplish any workload of project p in the current solution. Each spotted assignment (k,p) where worker k does not contribute to project p is immediately dropped, i.e.,

Algorithm 6.13 Subprocedure of Algorithm 6.12: Cancellation of assignments without contribution

```
1: for all (k, p) ∈ C do
2:     if Σ_{t∈T_p} f^proj_kpt = 0 then          // If Σ_{s∈S^match_kp} Σ_{t∈T_p} y_kpst = 0
3:         x_kp := 0
4:         C := C \ {(k, p)}
5:         for all t ∈ T_p do
6:             Update the remaining linear program LP^rem_t;     // Require f^proj_kpt = 0
```

x_{kp} is set to 0. As x_{kp} is finally fixed now, the pair (k, p) is removed from $\mathcal{C}$. Algorithm 6.13 runs in $O(KPT)$ time.

The concept that we applied in Algorithm 6.13 has also been applied by Di Gaspero et al. (2007). They have outlined a drop method for the problem of finding the minimum number of shifts that are required to meet staff requirements in a firm. In their drop method, which is called GreedyMCMF, they drop one shift at a time and solve a minimum cost maximum flow problem to check if a drop operation is feasible. The underlying flow network contains for each shift an edge. All shifts to which no worker is assigned, i.e., all corresponding edges with zero flow, are immediately dropped whenever a solution has been determined.

The second extension, which can also reduce the number of pairs (k, p) in the list $\mathcal{C}$ in the course of DROP, comprises two lower bounds. In the following, we will describe these two bounds and how they can be utilized to remove assignments from the list $\mathcal{C}$ without the need to check feasibility by solving the remaining linear program.

The first lower bound is denoted by LB_p^{Drop}, $p \in \mathcal{P}$. It is a lower bound on the number of assignments of workers to project p that are needed to cover all requirements of project p. The bound LB_p^{Drop} is a local lower bound, as for its computation only workers who are currently assigned to project p are considered. The computation of the lower bound LB_p^{Drop} is similar to the computation of the local version of the global lower bound LB_p^{glob}, which was outlined in Algorithm 6.3 on page 118.

The lower bound LB_p^{Drop} can be used to remove assignments from the list $\mathcal{C}$. If in a current solution $\boldsymbol{x}$ exactly LB_p^{Drop} workers are assigned to project p, dropping any further assignment $(k, p') \in \mathcal{C}$ with $p' = p$ will lead to infeasibility of the remaining LP. Hence, we can remove all assignments that contain project p from the list $\mathcal{C}$; the corresponding binary variables can be finally fixed to 1.

Algorithm 6.14 describes how the lower bound LB_p^{Drop} for project p is calculated and how it is utilized to remove those pairs (k, p) from the list $\mathcal{C}$ for which the corresponding variable x_{kp} must not be set to 0, because setting x_{kp} to 0 would render the current solution $\boldsymbol{x}$ infeasible. Algorithm 6.14 requires $O\left(K(S^2 + TS + K + P)\right)$ time if quicksort is used for sorting operations.

Within the heuristic DROP, Algorithm 6.14 is called several times and the value of LB_p^{Drop} can change. In the preparatory part of DROP, Algorithm 6.14 is called for all projects $p \in \mathcal{P}$ after an initial solution for $\boldsymbol{y}$ has been calculated. In the main part of DROP, Algorithm 6.14 is called for a project p whenever a drop operation associated with project p succeeded, because only when dropping a pair (k, p), $k \in \mathcal{K}_p^{\text{suit}}$, succeeded, the

Algorithm 6.14 Subprocedure of Algorithm 6.12: Calculation of the lower bound LB_p^{Drop} on the number of assignments of workers to project p

Input: Project p

$LB_p^{\text{Drop}} := 0$;
Declare a vector *vecProjCap* of length $\sum_{k \in \mathcal{K}_p^{\text{suit}}} x_{kp}$;
for all $t \in \mathcal{T}_p$ **do**
  $r_{pt} := \sum_{s \in S_p} r_{pst}$;
  $i := 1$;
  for all $k \in \mathcal{K}_p^{\text{suit}} \mid x_{kp} = 1$ **do**
    $R_{kt}^{\text{rem}} := R_{kt}$;
    $vecProjCap[i] := 0$;
    for all $s \in \mathcal{S}_{kp}^{\text{match}}$ in order of non-increasing skill levels l_{ks} **do**
      if $R_{kt}^{\text{rem}} > 0$ **then**
        $maxHours := \min(r_{pst}, R_{kt}^{\text{rem}} l_{ks})$;
        $vecProjCap[i] := vecProjCap[i] + maxHours$;
        $R_{kt}^{\text{rem}} := R_{kt}^{\text{rem}} - \frac{maxHours}{l_{ks}}$;
    $i := i + 1$;
  Sort the entries in *vecProjCap* in order of non-increasing values;
  $n^* = \min\{n \in \mathbb{N} \mid r_{pt} - \sum_{i=1}^{n} vecProjCap[i] \leq 0\}$;
  $LB_p^{\text{Drop}} := \max(LB_p^{\text{Drop}}, n^*)$;
if $LB_p^{\text{Drop}} = \sum_{k \in \mathcal{K}_p^{\text{suit}}} x_{kp}$ **then**
  for all $(k, p') \in \mathcal{C}$ **do**
    if $p' = p$ **then**
      $\mathcal{C} := \mathcal{C} \setminus \{(k, p')\}$;

value of LB_p^{Drop} can change. The value of LB_p^{Drop} monotonically increases in the course of DROP.

The second lower bound is denoted by LB_{ps}^{Drop}, $p \in \mathcal{P}$, $s \in \mathcal{S}_p$. It is a lower bound on the number of workers mastering skill s that must be assigned to project p in order to satisfy all requirements of project p for skill s. The bound LB_{ps}^{Drop} is also a local lower bound, as for its computation only workers who are currently assigned to project p are considered. The computation of the lower bound LB_{ps}^{Drop} resembles the computation of the local lower bound LB_{ps}^{loc}, which was outlined in Algorithm 6.2 on page 116.

The lower bound LB_{ps}^{Drop} can be exploited to clear assignments from the list $\mathcal{C}$. If in a current solution $\boldsymbol{x}$ exactly LB_{ps}^{Drop} workers who master skill s are assigned to project p, dropping any further assignment $(k, p') \in \mathcal{C}$ with $p' = p$ and $k \in \mathcal{K}_s$ will lead to infeasibility of the remaining LP. Thus, we can eliminate all these assignments from $\mathcal{C}$; the corresponding binary variables can be finally fixed to 1.

Algorithm 6.15 outlines how the lower bound LB_{ps}^{Drop} for project p and skill s is computed and how it is used to erase those pairs (k, p) from the list $\mathcal{C}$ for which the corresponding variable x_{kp} must not be set to 0, because setting x_{kp} to 0 would render the

current solution $\boldsymbol{x}$ infeasible. Algorithm 6.15 runs in $O(TK^2 + KP)$ time if quicksort is used for sorting operations.

Algorithm 6.15 Subprocedure of Algorithm 6.12: Calculation of the lower bound LB_{ps}^{Drop} on the number of assignments of workers to project p for skill s

1: **Input:** Project p, skill $s \in \mathcal{S}_p$
2:
3: $LB_{ps}^{\text{DROP}} := 0$;
4: Declare a vector *vecSkillCap* of length $\sum_{k \in \mathcal{K}_s} x_{kp}$;
5: **for all** $t \in \mathcal{T}_p$ **do**
6: $\quad r_{pst}^{\text{rem}} := r_{pst}$;
7: $\quad i := 1$;
8: $\quad$ **for all** $k \in \mathcal{K}_s \mid x_{kp} = 1$ **do**
9: $\quad\quad$ *vecSkillCap*$[i] := R_{kt} l_{ks}$;
10: $\quad\quad i := i + 1$;
11: $\quad$ Sort the entries in *vecSkillCap* in order of non-increasing values;
12: $\quad n^* := \min\left\{n \in \mathbb{N} \mid r_{pst}^{\text{rem}} - \sum_{i=1}^{n} \textit{vecSkillCap}[i] \leq 0\right\}$;
13: $\quad LB_{ps}^{\text{Drop}} := \max(LB_{ps}^{\text{Drop}}, n^*)$;
14: **if** $LB_{ps}^{\text{Drop}} = \sum_{k \in \mathcal{K}_s} x_{kp}$ **then**
15: $\quad$ **for all** $(k, p') \in \mathcal{C}$ **do**
16: $\quad\quad$ **if** $p' = p$ **and** $k \in \mathcal{K}_s$ **then**
17: $\quad\quad\quad \mathcal{C} := \mathcal{C} \setminus \{(k, p')\}$;

Algorithm 6.15 is called repeatedly within the heuristic DROP and the value of LB_{ps}^{Drop} can change in the course of DROP. In the preparatory part of DROP, Algorithm 6.15 is called for each project $p \in \mathcal{P}$ and for each skill $s \in \mathcal{S}_p$ required by project p, after an initial solution for $\boldsymbol{y}$ has been computed. In the main part of DROP, Algorithm 6.15 is called whenever a drop operation for a pair (k, p) succeeded. Then, Algorithm 6.15 is called for each skill s that is required by project p and mastered by worker k. Only when dropping a pair (k, p), $k \in \mathcal{K}_s$, succeeded, the value of LB_{ps}^{Drop} can change. The value of LB_{ps}^{Drop} monotonically increases in the course of DROP.

Algorithms 6.14 and 6.15 were implemented in a more efficient way than in the way presented here for reasons of clarity. In our actual implementation, static values such as r_{pt} in line 6 of Algorithm 6.14 and entries in the vector *vecSkillCap* in line 9 of Algorithm 6.15 are not computed every time the corresponding algorithm is called, but only once.

For a multi-start version, we embedded DROP in the generic multi-start procedure outlined in Algorithm 6.5 on page 126. This multi-start version of DROP constructs different solutions, because assignments to cancel are randomly selected within DROP. However, we do not build a solution from scratch in each pass of the multi-start method. This means that we do not start with a solution where each worker is assigned to every suitable project. To save time, we execute Algorithm 6.11, the preparatory part of DROP, before the multi-start procedure outlined in Algorithm 6.5 is started. In the preparatory part, the remaining linear programs are solved once for each period. Then, assignments without contribution are canceled and further assignments are canceled for each project p if indicated by the bounds LB_{p}^{Drop} and LB_{ps}^{Drop}, $s \in \mathcal{S}_p$. The resulting solution is stored

and used as the starting point for each pass of the multi-start method. Hence, in each pass, some variables x_{kp} are kept fixed to the same values as in all previous passes.

The fact that some variables remain fixed during the multi-start procedure constitutes a small degree of rebuild according to Martí et al. (2010, p. 274). The actual idea behind a certain degree of rebuild is to start from a good starting point and not to save time. In our case, starting each pass with some variables already fixed did save time, but also raised the question whether solution quality would suffer, because fixing some variables to the same values in each pass narrows the solution space that is explored. Preliminary tests revealed that solution quality suffered, but the tests also revealed a substantial gain in solution time, which justifies the approach in our opinion.

So far, we suggested to solve the remaining linear programs $\mathrm{LP}_t^{\mathrm{rem}}$, $t \in \mathcal{T}$, by the standard simplex method using a solver from the package CPLEX. In the following two subsections, we will show how the remaining linear programs $\mathrm{LP}_t^{\mathrm{rem}}$, $t \in \mathcal{T}$, can be solved more efficiently by two means. One mean is to apply the generalized network simplex method, because the remaining linear programs can be formulated as generalized maximum flow problems. The generalized network simplex method is more efficient than the standard simplex method for generalized network flow problems. Another mean is to reformulate the remaining linear programs as generalized minimum cost flow problems in order to facilitate warm starts of the generalized network simplex method. From these two means, we first consider the reformulation of a remaining LP as a generalized minimum cost flow problem in the next Subsection 6.2.3.2. Thereafter, we will outline the generalized network simplex method in Subsection 6.2.3.3.

6.2.3.2 Representation of a remaining LP as a generalized minimum cost flow problem

In this subsection, we will formulate the remaining linear program $\mathrm{LP}_t^{\mathrm{rem}}$, $t \in \mathcal{T}$, which must be solved within DROP, as a generalized maximum flow problem and then reformulate this problem as a generalized minimum cost flow problem. We will demonstrate how the reformulation can accelerate the DROP procedure.

To prepare the formulation of a remaining linear program $\mathrm{LP}_t^{\mathrm{rem}}$ as a generalized minimum cost flow problem, we will first show that $\mathrm{LP}_t^{\mathrm{rem}}$ can be formulated as a generalized maximum flow problem. The term *generalized* indicates that the corresponding flow problem arises for a network with gains. Both the minimum cost flow and the maximum flow formulation require that the network model, which we presented in Subsection 4.3.1, is used as the underlying formulation of the workforce assignment problem. Fixing all assignment variables x_{kp} to either 0 or 1 leads to a linear program $\mathrm{LP}_t^{\mathrm{rem}}$ for each period $t \in \mathcal{T}$.

To see that a remaining linear program $\mathrm{LP}_t^{\mathrm{rem}}$ that originates from the network model can be transformed into a generalized maximum flow problem, consider Figure 4.1 on page 61. In Figure 4.1, we sketched a section of the network model for a single period t. A solution that satisfies all constraints for period t requires a sufficiently large flow emanating from the workers. This flow must satisfy the demands of projects and departments. The complete model, which is only indicated in Figure 4.1, can be interpreted as a generalized maximum flow problem. To make this interpretation obvious and to facilitate the use of computational methods, we slightly modify the network flow model that is sketched in

Figure 4.1. We introduce a source node and a terminal node that are connected to nodes at stage 1 and at stage 4, respectively. Additionally, we replace supplies and demands of nodes by arc capacities, i.e., by upper flow bounds. Details of these modifications are described in the following two paragraphs. We call the resulting network maximum flow network.

To obtain the maximum flow network, we first introduce a source node, which constitutes stage 0. From this source node, an arc runs to each node at stage 1. The capacity of such an arc running to the node of worker $k \in \mathcal{K}$ at stage 1 is set to the availability R_{kt} of worker k. The supply of all nodes at stage 1 is set to 0.

An almost mirror-image modification is applied to the opposite side of the network depicted in Figure 4.1. We introduce a terminal node, which constitutes stage 5, and link every node at stage 4 to this terminal node. Recall that each node at stage 4 is a demand node. The upper flow bound of an arc that runs from a node at stage 4 to the terminal node at stage 5 is set to the demand of the from-node. In case of a project demand node that is associated with project p and skill s, the upper flow bound is set to r_{pst} and in case of a demand node that is associated with department d, the upper flow bound is set to rd_{dt} . The demands of all nodes at stage 4 are set to 0.

To solve the remaining linear program $\mathrm{LP}_t^{\mathrm{rem}}$ that corresponds to the maximum flow network, we determine a maximum flow from the new source node to the new terminal node. For this determination, we implicitly assume that the source node has an infinite supply and that the terminal node has an infinite demand. A feasible solution for $\mathrm{LP}_t^{\mathrm{rem}}$ exists if there is a maximum flow that saturates all arcs running from stage 4 to stage 5 and that observes the capacities of all arcs running from stage 0 to stage 1. Such a flow corresponds to a solution where the requirements of all projects and all departments in period t are satisfied and where the availabilities of all workers are not exceeded. Hence, we can solve a generalized maximum flow problem to decide whether a feasible solution exists for $\mathrm{LP}_t^{\mathrm{rem}}$ or not.

The transformation of the maximum flow problem that is associated with $\mathrm{LP}_t^{\mathrm{rem}}$, $t \in \mathcal{T}$, into a minimum cost flow problem is not difficult. Ahuja et al. (1993, pp. 6–7) describe how to formulate a maximum flow problem as a minimum cost flow problem. Figure 6.4 illustrates how the generalized minimum cost flow network of a period t looks like for an example with two workers k_1 and k_2. Worker k_1 masters the skills s_1 and s_2, worker k_2 masters skill s_1. Both workers can contribute to the projects p_1 and p_2, where p_1 requires the skills s_1 and s_2 and p_2 requires only s_1. The network is a directed graph $G = (N, A)$ with node set N and arc set A. As the network for the generalized maximum flow problem, the network for the generalized minimum cost flow problem features a source node (node 0, stage 0) and a terminal node (node 18, stage 5). In addition, there is an arc running from the terminal node to the source node. This backward arc induces a cycle. And, there are two self-loops running from node 0 to node 0. They are also network arcs.

The network arcs $\langle i, j\rangle \in A$ feature upper flow bounds u_{ij}, costs c_{ij}, and gains μ_{ij}. For all arcs $\langle i, j\rangle \in A$, we set the upper bound $u_{ij} := +\infty$, the costs $c_{ij} := 0$, and the gain $\mu_{ij} := 1$ if we do not explicitly mention other values in the following paragraphs. The parameters whose values deviate from the default values will be first described for the backward arc and the two self-loops, then for the other arcs.

The backward arc, which runs from the terminal node to the source node, has infinite capacity. Its cost are -1 making this arc the only arc with non-zero costs in the network.

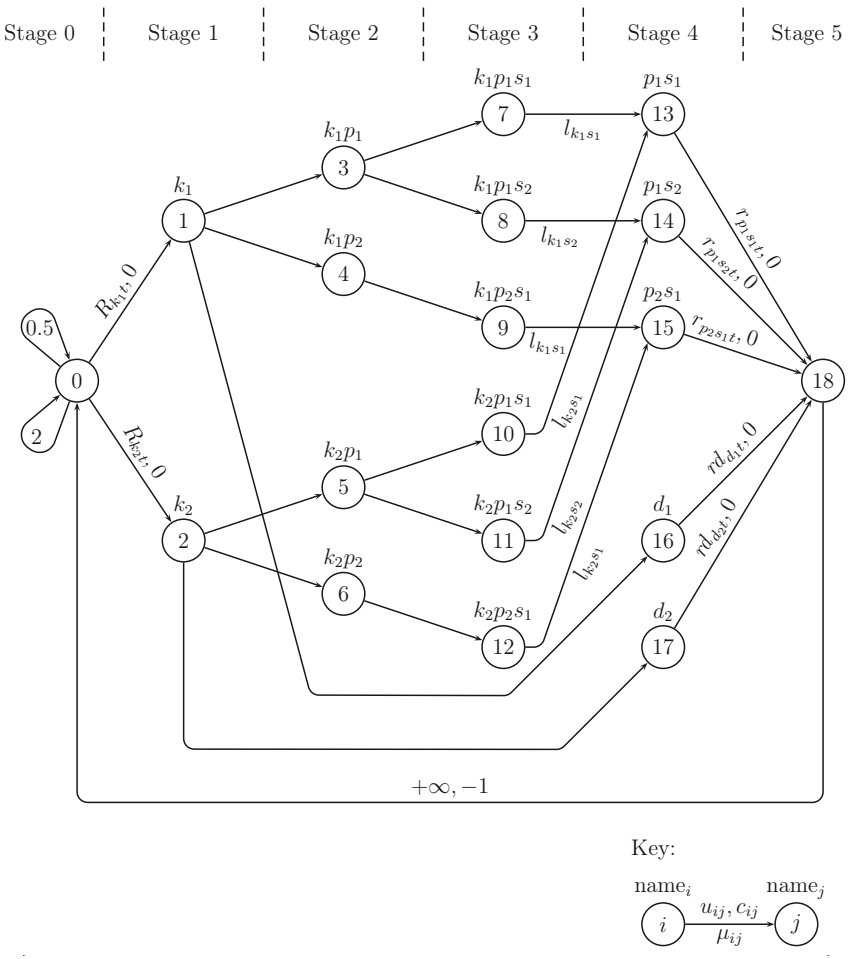


Figure 6.4: Example for a generalized minimum cost flow network representing a remaining linear program LP_t^{rem} (if not indicated otherwise, then upper flow bounds $u_{ij} := +\infty$, costs $c_{ij} := 0$ and gains $\mu_{ij} := 1$; index t has been omitted for node names)

The backward arc allows flow to circulate through the network; hence we do not have to assume infinite supply and demand for the source node and the terminal node, respectively.

Out of the two self-loops that were added to node 0, one is a gainy self-loop (gain $\mu_{00} = 2$) which can generate flow and the other one is a lossy self-loop (gain $\mu_{00} = 0.5$)

which can absorb excessive flow.[5] The self-loops are required to balance the network flow, as they enable flow conservation constraints to hold for each node $i \in N$. Flow imbalances can occur due to the gains in the network.

For each arc running from node 0 at stage 0 to node $k \in \mathcal{K}$ at stage 1, we set the upper bound $u_{0k} := R_{kt}$, which represents the limited capacity of worker k in period t.

Stage 4 of the network in Figure 6.4 comprises those nodes that represent the demands of projects and departments. A project demand node at stage 4 that corresponds to project $p \in \mathcal{P}$ and skill $s \in \mathcal{S}_p$ represents the requirement r_{pst}. A departmental demand node at stage 4 that corresponds to department $d \in \mathcal{D}$ represents the requirement rd_{dt}. For each arc running from a project demand node or a departmental demand node at stage 4 to the terminal node 18 at stage 5, the upper bound was set to the demand r_{pst} or rd_{dt}, respectively.

The flow on each arc emanating from a node at stage 3 and going to a node at stage 4 represents the time spent by worker $k \in \mathcal{K}$ for project $p \in \mathcal{P}_k^{\text{suit}}(t)$ on skill $s \in \mathcal{S}_{kp}^{\text{match}}$. Hence, we set the gain of these arcs to the associated skill levels l_{ks}.

Note that a minimum cost flow in the outlined network contains a maximum flow from the source node to the terminal node. This coincidence of maximum flow and minimum cost flow results from the special cost structure of the network. A minimum cost flow is reached in this network when the maximum number of flow units circulates through the network and hence passes the backward arc, which generates cost savings due to its costs of -1.

Compared to the generalized maximum flow problem, the generalized minimum cost flow problem has the advantage for our application that warm starts are possible. Assume that a feasible flow f in a period t is known for a given assignment $\boldsymbol{x}$ of workers to projects and let the flow from stage 1 to stage 2 on the arc $\langle k', p' \rangle$ between worker k' and project p' in period t be greater than 0, i.e., let $f_{k'p'} > 0$. If $x_{k'p'}$ is switched from 1 to 0 to obtain a new solution $\boldsymbol{x}'$, the flow f becomes infeasible for period t in case of a maximum flow problem, because we have to set the upper bound of arc $\langle k', p' \rangle$ to 0. In case of a minimum cost flow problem, though, only the costs $c_{k'p'}$ of arc $\langle k', p' \rangle$ are increased from zero to a (prohibitively) high positive value and the flow f is still feasible, even though not optimal in general. To check feasibility of the new solution $\boldsymbol{x}'$ with $x_{k'p'} = 0$, we must find a flow f' with $f'_{ij} = 0$ for all arcs $\langle i, j \rangle$ between stage 1 and stage 2 with $c_{ij} > 0$. The existence of such a flow can be checked by searching for a minimum cost flow in the generalized minimum cost flow network with increased costs $c_{k'p'}$. The search can be started from the flow f, since f is a primal feasible solution for the modified network.

The advantage of warm starts can be exploited many times. If we consider a remaining linear program LP_t^{rem} as a generalized minimum cost flow problem, a feasible flow f for LP_t^{rem} is always a primal feasible solution for a remaining linear program $\text{LP}_t^{\text{rem}'}$ that arises in a succeeding drop operation and affects the same period t. Thus, the flow f can be used as a start solution for a succeeding LP. In the course of the heuristic DROP, many succeeding LPs must be solved. A warm start can be made many times. Hence, the formulation of the remaining LPs as generalized minimum cost flow problems may considerably accelerate DROP.

Kolisch and Heimerl (2012) have exploited this advantage of a generalized minimum

[5] The gains of the self-loops and the self-loops themselves are distinguished by their respective arc numbers, which will be introduced in Subsection 6.2.3.3.

cost flow formulation in the same way when repeatedly solving a staffing subproblem. To solve a single staffing subproblem they used the generalized network simplex method. We also applied the generalized network simplex method to solve our generalized minimum cost flow problems. We will describe this method and our implementation of this method in the next subsection.

6.2.3.3 Implementation of the generalized network simplex method

In this subsection, we will outline the generalized network simplex method and show how it can be used to efficiently check if a feasible solution exists for a remaining linear program $\mathrm{LP}_t^{\mathrm{rem}}$ that must be solved in the course of DROP. During our description of the generalized network simplex method we will introduce some basic definitions and data structures from graph theory when needed to understand the description. Although the generalized network simplex is a well-known solution method, this description is valuable for two reasons. First, it explains how the generalized network simplex can be applied to solve our subproblems $\mathrm{LP}_t^{\mathrm{rem}}$. Second, the description provides detailed pseudo code of key procedures of this solution method. The pseudo code allows the reader to easily implement the procedures himself. This subsection will be complemented by a short discussion of our implementation in Subsection 6.2.3.4.

The generalized network simplex is a special version of Dantzig's simplex method adapted to network structures with gains. Dantzig (1963, Chapter 17) already outlined an adaptation to network structures without gains (see also Dantzig and Thapa, 1997, Section 9.8). Our implementation for the case with gains follows Ahuja et al. (1993, Chapter 15) and Bazaraa et al. (2010, Chapter 9).[6] Bazaraa et al. (2010, Chapter 9) have only *sketched* tree updates that are required at the end of every pivot operation. For these tree updates, we provide detailed pseudo code that has not been published so far to the best of our knowledge.

The network that we consider is a directed graph $G = (N, A)$ with node set N and arc set A. For each network arc $\langle i, j\rangle \in A$, a gain μ_{ij} is given. Each arc $\langle i, j\rangle \in A$ can also be identified by an arc number a. Then μ_{ij} can also be denoted by μ_a; the same holds for other arc quantities. If $\mu_{ij} < 1$, arc $\langle i, j\rangle$ is called lossy, if $\mu_{ij} > 1$, it is called gainy. The arc gains μ_{ij} on the arcs between stage 3 and 4 are used to represent the different skill levels l_{ks}, as illustrated in Figure 6.4 on page 159.

The *network simplex method* applies only to networks where the gain μ_{ij} is equal to 1 for each arc $\langle i, j\rangle$ of the network. The method moves from one basis solution to another, where all basis solutions correspond to spanning trees of G.

In contrast, for the *generalized network simplex method*, each basis solution corresponds to a collection of 1-trees. A 1-tree, also called augmented tree, is a spanning tree of a graph with one additional arc $\langle \alpha, \beta\rangle \in A$, called *extra arc*, hence, a 1-tree contains exactly one cycle. Because $\alpha = \beta$ is possible, the cycle may be a self-loop. A collection of 1-trees is called augmented forest.[7] Each 1-tree tr of an augmented forest, which represents a basis solution, spans a subgraph $G^{tr} = (N^{tr}, A^{tr})$ of $G = (N, A)$ with $N^{tr} \subseteq N$ and $A^{tr} \subset A$, where A^{tr} does not include the extra arc $\langle \alpha, \beta\rangle$. The graph G^{tr} and the extra

[6] For an alternative presentation of the generalized network simplex method, see Jensen and Barnes (1987, Chapters 9 and 10).

[7] Confer Ahuja et al. (1993, pp. 574–576). A forest is a collection of trees (cf. Jungnickel, 2005, p. 8).

arc $\langle \alpha, \beta \rangle$ constitute the 1-tree tr. Since $\alpha \in N^{tr}$ and $\beta \in N^{tr}$, the extra arc induces a cycle. If the augmented forest consists of two or more 1-trees, for each pair (tr, tr') of 1-trees, $N^{tr} \cap N^{tr'} = \emptyset$ holds, while the union of all the nodes of all 1-trees coincides with N. Actually, a basis solution is represented by *good 1-trees* which are 1-trees with a lossy or gainy cycle. These 1-trees constitute a *good augmented forest* (cf. Ahuja et al., 1993, pp. 574–576).

Our implementation draws upon data structures from graph theory. We will briefly introduce these data structures before they are used throughout the remainder of this subsection. To represent a 1-tree, we fall back to the representation of a spanning tree and additionally save information about the extra arc. We save the number of the extra arc, a^{extra}, its from-node α, and its to-node β. For a spanning tree, usually one node is specified as root node. Likewise, we specify one node of a 1-tree tr as the root node r_{tr}. We chose the root node to be incident with the extra arc a^{extra}_{tr}, i.e., $r_{tr} := \alpha$ or $r_{tr} := \beta$, hence, the root node is part of the cycle. We used eight node-length vectors to store information about a 1-tree.[8] Fewer vectors would have been sufficient, but the more information stored, the faster some operations can be executed, though at the expense of higher memory demand. However, in practice, memory capacity is not a bottleneck, whereas time must be considered scarce. The eight node-length tree indices are:

(1) **Predecessor index** (pred(i), $i \in N^{tr}$): For each node $i \in N^{tr} \setminus \{r_{tr}\}$, there is a unique path from i to the root node r_{tr}. The immediate predecessor node of node i on this path is saved as pred(i). For the root node, we set pred(r_{tr}) $:= -1$.

(2) **Depth index** (depth(i), $i \in N^{tr}$): The depth index of node i records the number of arcs in the path from i to the root node. It is also called distance index.

(3) **Thread index** (thread(i), $i \in N^{tr}$): The thread indices allow to traverse a 1-tree in a depth-first fashion visiting all nodes, starting from and returning to the root node. They also allow to visit all successors of an arbitrary node $i \in N^{tr}$. The successors of node i are those nodes in the 1-tree that are joined via node i to the root node r_{tr}. Node i and its successors constitute the subtree $tr(i)$. Starting from the thread index of an arbitrary node $i \in N^{tr}$, all successors of node i can be visited following the thread indices until the first node j is reached with depth(j) $\leq$ depth(i), i.e., the first node $j \notin N^{tr(i)}$. Note that usually the thread traversal is not unique.

(4) **Reverse thread index** (revThread(i), $i \in N^{tr}$): The reverse thread indices allow to traverse a tree in reversed order compared to the thread traversal. If thread(i) $= j$, then revThread(j) $= i$. During the reverse thread traversal, the next visited node is always a leaf node of the tree which is constituted by those nodes that have not been visited so far.

(5) **Final node index** (final(i), $i \in N^{tr}$): The index final(i) stores the successor of node i that is the last one visited during thread traversal of $tr(i)$.

(6) **Number of subtree nodes index** (SubtreeNodes(i), $i \in N^{tr}$): The index SubtreeNodes(i) saves the number of nodes belonging to the subtree $tr(i)$. This number is the number of successors of node i increased by 1.

[8] Some additional node-length vectors were used for saving further information that does not refer to spanning trees but to 1-trees only. These vectors will be introduced later on (see page 167).

(7) **Arc number and orientation index** (ArcAndOrient(i), $i \in N^{tr}$): For each node $i \in N^{tr} \setminus \{r_{tr}\}$, the index ArcAndOrient(i) saves the arc number of the arc linking node i and its predecessor node pred(i). This arc is either the arc $\langle \text{pred}(i), i \rangle$ or the arc $\langle i, \text{pred}(i) \rangle$. For the root node, the index ArcAndOrient(r_{tr}) is set to the number of the extra arc. The sign of the index is used to specify the orientation of the arc. If the arc ArcAndOrient(i) points from the root node away, i.e., in case of an arc $\langle \text{pred}(i), i \rangle$ (outbound arc), the sign of the index is positive. If the arc ArcAndOrient(i) points in direction of the root node, i.e., in case of an arc $\langle i, \text{pred}(i) \rangle$ (inbound arc), the sign of the index is negative.

(8) **Root node index** (Root(i), $i \in N^{tr}$): The root node index records for a node i the root node of the 1-tree or tree to which node i currently belongs. This index is relevant if a tree is split into two trees or, more generally speaking, if a set of trees or 1-trees is considered. Through this index not only the root node can be identified, but also the 1-tree itself if a 1-tree is identified by the number of its root node.

Example 6.9 For an illustrative example for the eight types of indices, consider the spanning tree given in Figure 6.5 with 14 nodes and 13 arcs. The corresponding indices are stated for two different choices of the root node r_{tr} in Table 6.2. The indices are given for $r_{tr} = 0$ in the upper half and for $r_{tr} = 8$ in the lower half of Table 6.2. □

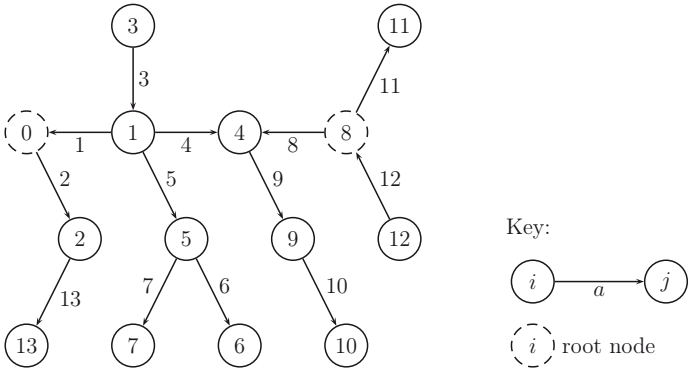


Figure 6.5: Example for a spanning tree; the tree indices for $r_{tr} = 0$ and for $r_{tr} = 8$ are given in Table 6.2

An initial feasible solution is required to start the generalized network simplex. Such an initial feasible solution, which is an initial good augmented forest, could be obtained by adding artificial self-loops with high costs c_{ii} to each node i in the initial network except for the source node 0, which has got two self-loops already (cf. Ahuja et al., 1993, p. 584). Then, each node forms a 1-tree with its self-loop representing the extra arc, while for each basic (and nonbasic) arc the arc flow is 0. However, this is a poor start solution, because all artificial self-loops do not belong to the basis of a solution that is feasible for

Table 6.2: Tree indices for the spanning tree given in Figure 6.5: in the upper half for $r_{tr} = 0$, in the lower half for $r_{tr} = 8$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
							$r_{tr} = 0$							
pred(i)	−1	0	0	1	1	1	5	5	4	4	9	8	8	2
depth(i)	0	1	1	2	2	2	3	3	3	3	4	4	4	2
thread(i)	1	3	13	4	8	6	7	2	11	10	5	12	9	0
revThread(i)	13	0	7	1	3	10	5	6	4	12	9	8	11	2
final(i)	13	7	13	3	10	7	6	7	12	10	10	11	12	13
SubtreeNodes(i)	14	11	2	1	6	3	1	1	3	2	1	1	1	1
ArcAndOrient(i)	–	−1	2	−3	4	5	6	7	−8	9	10	11	−12	13
Root(i)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
							$r_{tr} = 8$							
pred(i)	1	4	0	1	8	1	5	5	−1	4	9	8	8	2
depth(i)	3	2	4	3	1	3	4	4	0	2	3	1	1	5
thread(i)	2	3	13	5	9	6	7	0	11	10	1	12	4	8
revThread(i)	7	10	0	1	12	3	5	6	13	4	9	8	11	2
final(i)	13	13	13	3	13	7	6	7	13	10	10	11	12	13
SubtreeNodes(i)	3	8	2	1	11	3	1	1	14	2	1	1	1	1
ArcAndOrient(i)	1	−4	2	−3	8	5	6	7	–	9	10	11	−12	13
Root(i)	8	8	8	8	8	8	8	8	8	8	8	8	8	8

the original network, which does not feature artificial self-loops. Hence, many degenerate pivot operations are necessary to transfer the artificial self-loops to the set of nonbasic arcs.

In our case, though, it is possible to construct a better initial feasible good augmented forest. Starting with an arc flow of 0 for every arc, too, we build one initial 1-tree containing all network nodes. To build the initial 1-tree, we conduct a depth-first search that commences from the root node 0. The gainy self-loop of node 0 is stored as the extra arc of the initial 1-tree. For the network illustrated in Figure 6.4 on page 159, the resulting initial good augmented forest is depicted in Figure 6.6.

All arcs that belong to the initial 1-tree including the extra arc constitute the initial set of basic arcs, while all other arcs in A belong to the initial set of nonbasic arcs. Nonbasic arcs carry flow at their lower or upper bound. At the outset, the flow on every basic and nonbasic arcs is at its lower bound and thus is zero.

For a pivot operation of the generalized network simplex method, we add a nonbasic arc to the set of basic arcs and remove an arc from the resulting set of basic arcs. The latter arc is transferred to the set of nonbasic arcs. Each pivot operation is a move from one basis solution to another one that is as good as or better than the one before. We will summarize such a move in the remainder of this paragraph before we elaborate on intermediate steps of this move in the following paragraphs. For a pivot operation, a nonbasic arc which violates its optimality condition is selected. Let $\langle k, l\rangle$ denote this arc, which enters the basis. If the current flow on $\langle k, l\rangle$ is at its lower (upper) bound, we increase (decrease) the flow on $\langle k, l\rangle$ and propagate this change through the augmented

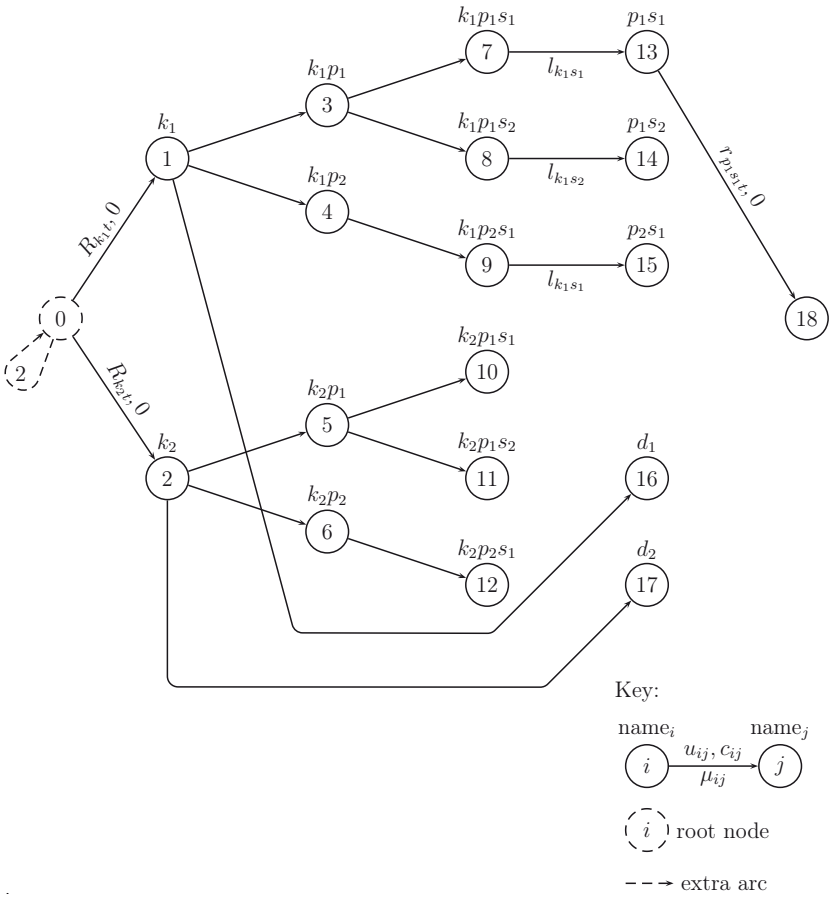


Figure 6.6: Initial good augmented forest derived from the network in Figure 6.4 on page 159 (if not indicated otherwise, then upper flow bounds $u_{ij} := +\infty$, costs $c_{ij} := 0$ and gains $\mu_{ij} := 1$; index t omitted for node names)

forest. The maximum absolute change δ of the flow on the entering arc is bounded by the flow bounds of arc $\langle k, l\rangle$ itself and by the flow bounds of other basic arcs. One arc bounding the increase or decrease of the flow on the entering arc is picked as the leaving arc $\langle p, q\rangle$, which leaves the basis. This arc $\langle p, q\rangle$ is added to the set of nonbasic arcs. In the following, we will describe the parts of a pivot operation in more detail.

The optimality condition of an arc $\langle i, j\rangle$ carrying flow at its lower (upper) bound is violated if its reduced costs $c_{ij}^{\text{red}} < 0$ ($c_{ij}^{\text{red}} > 0$), where the reduced costs are computed

from the dual variables π_i, $i \in N$, as follows: $c_{ij}^{\text{red}} = c_{ij} - \pi_i + \mu_{ij}\pi_j$ (cf. Ahuja et al., 1993, p. 577).

The dual variables π_i, $i \in N$, also called node potentials, can be calculated exploiting that $c_{ij}^{\text{red}} = 0$ holds for all basic arcs $\langle i, j\rangle$. Algorithm 6.16 is a description in pseudo code of the algorithm we implemented to compute the node potentials. It is based on the procedure outlined in Ahuja et al. (1993, p. 578) to compute the node potentials of a 1-tree tr. Demanding $c_{ij}^{\text{red}} = c_{ij} - \pi_i + \mu_{ij}\pi_j = 0$ to hold for every arc $\langle i, j\rangle$ in the 1-tree tr leads to a system of linear equations in π_i, $i \in N^{tr}$. To solve this linear system, Algorithm 6.16 sets the potential $\pi_{r_{tr}}$ of the root node r_{tr} equal to θ, i.e., $\pi_{r_{tr}} := \theta = f_{r_{tr}} + g_{r_{tr}}\theta$ with $f_{r_{tr}} := 0$ and $g_{r_{tr}} := 1$. The value of θ is unknown and must be determined. To this aim, the 1-tree is traversed following the thread indices. For each node i that is visited, the node potential π_i is described by the linear expression $\pi_i = f_i + g_i\theta$. The value of the coefficients f_i and g_i can be calculated from the equation $c_{ij} - \pi_i + \mu_{ij}\pi_j = c_{ij} - (f_i + g_i\theta) + \mu_{ij}(f_j + g_j\theta) = 0$. After the traversal, we calculate the value of θ considering the equation $c_{\alpha\beta} - \pi_\alpha + \mu_{\alpha\beta}\pi_\beta = 0$ that corresponds to the extra arc $\langle \alpha, \beta\rangle$, where the wheel comes full circle and the potential must be balanced. Knowing θ, the dual variables can be computed in a second thread traversal. Algorithm 6.16 runs in $O(|N^{tr}|)$ time.

Algorithm 6.16 Algorithm to compute the node potentials of a 1-tree

Input: 1-tree tr belonging to a basis solution
Output: π_i, $i \in N^{tr}$

$j := r_{tr}$ // Initialize j with the root node of 1-tree tr
$f_j := 0, \quad g_j := 1;$
$j := \text{thread}(j);$
while $j \neq r_{tr}$ **do**
    $i := \text{pred}(j);$
    $a := |\text{ArcAndOrientation}(j)|;$ // Get the arc number
    if $\text{ArcAndOrientation}(j) > 0$ **then** // If arc $\langle i, j\rangle \in A^{tr}$
        $f_j := \dfrac{f_i - c_a}{\mu_a}, \quad g_j := \dfrac{g_i}{\mu_a}$
    else // If arc $\langle j, i\rangle \in A^{tr}$
        $f_j := \mu_a f_i + c_a, \quad g_j := \mu_a g_i;$
    $j := \text{thread}(j);$
$\theta = \dfrac{c_{\alpha\beta} - f_\alpha + \mu_{\alpha\beta} f_\beta}{g_\alpha - \mu_{\alpha\beta} g_\beta};$
$i := r_{tr};$
do
    $\pi_i := f_i + g_i\,\theta;$
    $i := \text{thread}(i);$
while $i \neq r_{tr}$

Given the node potentials π_i, $i \in N$, we apply the candidate list pivoting rule (cf. Ahuja et al., 1993, p. 417) to select a nonbasic arc $\langle k, l\rangle$ as the entering arc. Our pivoting rule is based on a candidate list of nonbasic arcs violating their optimality condition. The

rule selects an arc with maximum violation among all arcs in the candidate list as the entering arc. At the outset and from time to time, the candidate list is reconstructed from scratch. For a reconstruction from scratch, the nonbasic arcs are considered in a wraparound fashion beginning with the nonbasic arcs emanating from that node i that would have been the next one in the previous reconstruction step. For a nonbasic arc, its reduced costs are computed and the arc is added to the candidate list if it violates its optimality condition. After having considered the outgoing arcs of node i, the outgoing arcs of the next nodes $i+1, i+2, \ldots$ are checked. Nodes are considered until the predefined capacity of the candidate list is exhausted or already exceeded or until node i, the starting point, is reached again. If the candidate list is empty after reconstruction, an optimal solution was found. If the candidate list is not empty, an arc in the list with maximum violation is selected as the entering arc $\langle k, l \rangle$.

The nodes k and l, which are incident with the entering arc, belong either to one 1-tree or to two different 1-trees of the augmented forest. Let us term the first case "one old 1-tree" and the second case "two old 1-trees". In both cases, the addition of arc $\langle k, l \rangle$ to the basis, i.e., to the good augmented forest, forms a so called bicycle. A bicycle is a spanning tree with two additional arcs. Thus, a bicycle contains exactly two cycles. The graph that is formed by the updated set of basic arcs contains a bicycle, as this graph now contains a subgraph with two cycles.

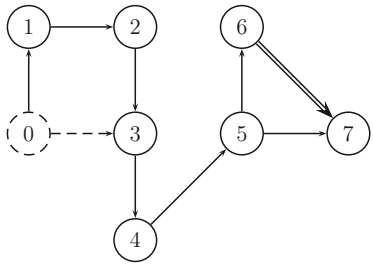
In the first case (one old 1-tree), two different types of bicycles can originate. An example for each type is given in Figure 6.7(a) and 6.7(b), respectively. The first type of a bicycle features two separate cycles (cf. Figure 6.7(a)). Note that separate cycles require that no node belongs to both cycles, i.e., both cycles are separated by a path that contains at least one arc. The second type of bicycle features two cycles that overlap. Overlapping cycles share one or more nodes (cf. Figure 6.7(b)).

In the second case (two old 1-trees), only one type of a bicycle originates. This type features separate cycles. An example of such a bicycle is given in Figure 6.7(c). Here, the two cycles cannot overlap.

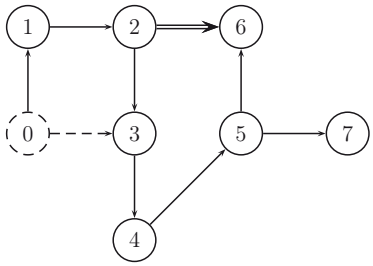
After having determined the entering arc and before determining the leaving arc, we identify those arcs on which the flow changes if the flow is changed on the entering arc $\langle k, l \rangle$. The change in flow on arc $\langle k, l \rangle$ affects the flow on those arcs that belong to a cycle of the bicycle which was created by $\langle k, l \rangle$. It also affects the flow on those arcs that belong to the path between the two cycles if there is such a path (cf. Ahuja et al., 1993, pp. 587–589). The flow does not change on all other arcs. Exploiting this information about arcs without flow changes helps to avoid unnecessary computations. Consequently, we identify those arcs of the bicycle that belong to a cycle or to the path between the two cycles if such a path exists. To be more precise, we do not identify those arcs directly, but identify the nodes that are incident with those arcs. Three node-length vectors are maintained that record information about cycle and path membership:

(1) **Old cycle membership index** (onOldCycle(i), $i \in N$): If node $i \in N$ belongs to the cycle of its 1-tree, the value of onOldCycle(i) is *true*, and *false* otherwise.

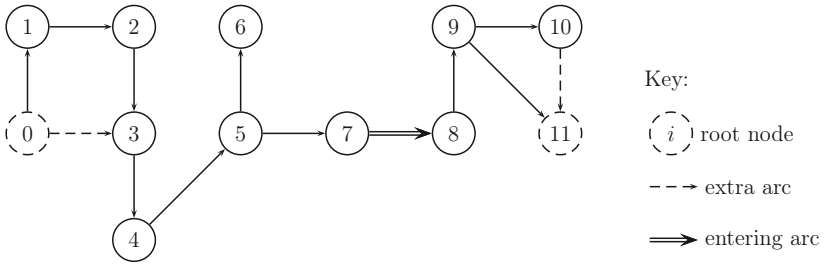
(2) **New cycle membership index** (onNewCycle(i), $i \in N$): If node $i \in N$ belongs to the new cycle that was induced in the current pivot operation by the entering arc, the value of onNewCycle(i) is *true*, and *false* otherwise.



(a) One old 1-tree, cycles do not overlap



(b) One old 1-tree, overlapping cycles



(c) Two old 1-trees

Figure 6.7: Examples for bicycles

(3) **Path membership index** (onPath(i), $i \in N$): If node $i \in N$ belongs to the path between the two cycles of the bicycle, the value of onPath(i) is *true*, and *false* otherwise.

To detect cycle membership and compute the indices onOldCycle(i) and onNewCycle(i), $i \in N$, we adapted the procedure given by Ahuja et al. (1993, p. 577, Figure 11.9).

Example 6.10 As an example for cycle and path membership, consider the bicycle in Figure 6.7(a): Nodes 0, 1, 2 and 3 belong to the old cycle; nodes 5, 6 and 7 belong to the new cycle; node 4 belongs to the path between both cycles. Since nodes 3 and 5 are part of a cycle, they are not counted as path nodes. In the bicycle depicted in Figure 6.7(c), all nodes belong to one of the two old cycles or to the path between these cycles, except node 6. Hence, the flow does not change on arc $\langle 5, 6\rangle$ if the flow is changed on the entering arc $\langle 7, 8\rangle$. □

In order to compute how the flow on the affected basic arcs changes when the flow on the entering arc is increased or decreased, we apply Algorithm 6.17 for the 1-trees that

form the bicycle. This means the algorithm is applied either once if one old 1-tree forms the bicycle, or twice if two old 1-trees form the bicycle. For a 1-tree tr, Algorithm 6.17 computes for each arc $\langle i,j\rangle \in A^{tr}$ the value $change_{ij}$ that indicates how the flow changes on arc $\langle i,j\rangle$ if the flow on the entering arc $\langle k,l\rangle$ is changed by one unit of flow ($change_{kl} := \pm 1$). Our algorithm follows the procedure sketched in Ahuja et al. (1993, pp. 585–586). To determine the flow changes $change_{ij}$, $\langle i,j\rangle \in A^{tr}$, an excess value $e(i)$ is introduced for each node i of the 1-tree. This excess value $e(i)$ is described by the linear expression $e(i) = f_i + g_i\theta$ depending on the unknown value of θ, which has to be computed. At the outset, Algorithm 6.17 sets the excess value of every node i to zero by setting $f_i := 0$ and $g_i := 0$.

If the flow on the entering arc $\langle k,l\rangle$ is increased, we assume an increase by one unit of flow ($change_{kl} := 1$) in Algorithm 6.17. The increase in flow leads to a negative excess (demand) of -1 at node k and to a positive excess (supply) of μ_{kl} at node l. If the flow on the entering arc $\langle k,l\rangle$ is decreased, we assume $change_{kl} := -1$ leading to analogous excesses at node k (supply) and node l (demand). This demand and this supply must be balanced by an arc flow within the bicycle. Note that a bicycle can generate and absorb flow by means of its gainy and lossy cycle, respectively.

To compute a balancing flow $change_{ij}$, $\langle i,j\rangle \in A^{tr}$, we set the flow on the extra arc $\langle \alpha,\beta\rangle$ of the 1-tree tr equal to θ by setting $change_{\alpha\beta} := \theta = F_{\alpha\beta} + G_{\alpha\beta}\theta$ with $F_{\alpha\beta} := 0$ and $G_{\alpha\beta} := 1$. The excess values $e(\alpha)$ and $e(\beta)$ are updated accordingly (lines 15 and 16 of Algorithm 6.17). Then, we traverse the 1-tree using the reverse thread index. During reverse thread traversal, each visited node is a leaf node of the 1-tree if we ignore nodes that have already been visited. For a visited leaf node j, its excess $e(j)$ is compensated by a flow on the arc which connects node j to its predecessor node $i = \text{pred}(j)$. After the traversal, the excess of the root node is compensated by choosing θ such that $e(r_{tr}) = f_{r_{tr}} + g_{r_{tr}}\theta = 0$ (line 28 of Algorithm 6.17). Knowing θ, the flow changes can be computed in a final traversal of the 1-tree tr. Algorithm 6.17 requires $O(|N^{tr}|)$ time.

Given the flow changes $change_{ij}$, we can identify the leaving arc denoted by $\langle p,q\rangle$ and update the flow values. The flow changes $change_{ij}$ allow us to compute the maximum absolute change δ for the flow on the entering arc for which one or more arc flows of the affected arcs reach their lower or upper bound. According to the value of δ, we update the flows on the arcs. One of those arcs for which the flow reaches a bound is selected as the leaving arc $\langle p,q\rangle$ which leaves the bicycle. Note that $\langle p,q\rangle = \langle k,l\rangle$ is possible.

Deleting the leaving arc from the bicycle leads to either one or in some cases to two new 1-trees that belong to the new basis. If, for example, in Figure 6.7(a) on the preceding page arc $\langle 1,2\rangle$ is the leaving arc, one new 1-tree will emerge. However, if arc $\langle 3,4\rangle$ is the leaving arc, for instance, two new 1-trees will emerge. For the bicycle in Figure 6.7(b), only one new 1-tree can emerge, because the leaving arc must be part of either one cycle or both cycles of the bicycle, as there is no path between them. One or two new 1-trees can arise again for the bicycle in Figure 6.7(c). Thus, depending on the type of bicycle and on the position of the leaving arc, various cases are possible for the number of emerging 1-trees. To distinguish these various cases, we exploit the records about cycle membership.

The indices describing the 1-tree(s) forming the bicycle must be updated after the leaving arc was deleted from the basis. Only if the leaving arc coincides with the entering arc, an update is not needed. As already noted, for some bicycle types, different cases are possible for the transition from the bicycle to either one or two new 1-trees. In

Algorithm 6.17 Algorithm to compute the changes of arc flows in a 1-tree caused by a unit flow change on the entering arc

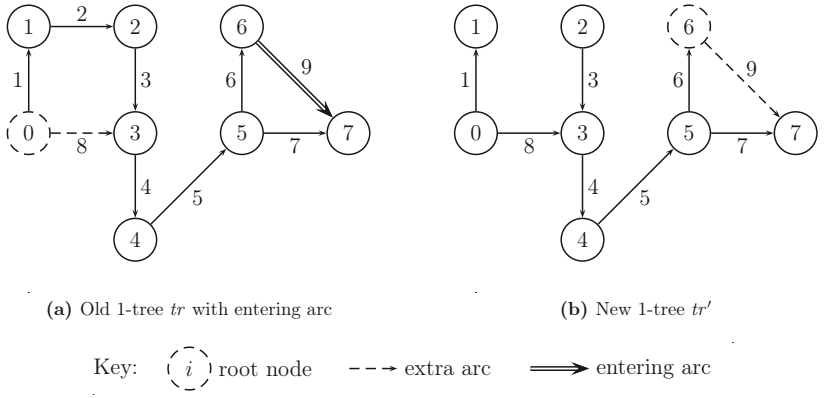
Input: Entering arc $\langle k, l\rangle$, 1-tree tr with $k \in N^{tr}$ or $l \in N^{tr}$
Output: $change_{ij}$, $\langle i, j\rangle \in A^{tr}$

for all $i \in N^{tr}$ **do**
    $f_i := 0, \quad g_i := 0;$
if arc $\langle k, l\rangle$ is at its lower bound **then**
    $change_{kl} := 1;$
    $f_k := -1, \quad g_k := 0;$
    $f_l := f_l + \mu_{kl}, \quad g_l := 0;$ // $k = l$ is possible
else // If arc $\langle k, l\rangle$ is at its upper bound
    $change_{kl} := -1;$
    $f_k := 1, \quad g_k := 0;$
    $f_l := f_l - \mu_{kl}, \quad g_l := 0;$ // $k = l$ is possible
$F_{\alpha\beta} := 0, \quad G_{\alpha\beta} := 1;$
$g_\alpha := g_\alpha - 1;$
$g_\beta := g_\beta + \mu_{\alpha\beta};$
$j :=$ revThread(r_{tr}); // Initialize j with a distinguished leaf node of the 1-tree tr
while $j \neq r_{tr}$ **do** // Visit all nodes $i \in N^{tr} \setminus \{r_{tr}\}$, trailing the reverse thread index
    if onOldCycle(j) = **true or** onNewCycle(j) = **true or** onPath(j) = **true then**
        $i :=$ pred(j);
        if arc $\langle i, j\rangle \in A^{tr}$ **then**
            $F_{ij} := -\dfrac{f_j}{\mu_{ij}}, \quad G_{ij} := -\dfrac{g_j}{\mu_{ij}};$
            $f_i := f_i + \dfrac{f_j}{\mu_{ij}}, \quad g_i := g_i + \dfrac{g_j}{\mu_{ij}};$
        else // If arc $\langle j, i\rangle \in A^{tr}$
            $F_{ji} := f_j, \quad G_{ji} := g_j;$
            $f_i := f_i + f_j\mu_{ji}, \quad g_i := g_i + g_j\mu_{ji};$
    $j :=$ revThread(j);
$\theta = -\dfrac{f_{r_{tr}}}{g_{r_{tr}}};$ // The root node r_{tr} is either α or β
$i := r_{tr};$
do
    if onOldCycle(i) = **true or** onNewCycle(i) = **true or** onPath(i) = **true then**
        $a :=$ |ArcAndOrientation(i)|; // Get the arc connecting node i and pred(i)
        $change_a := F_a + G_a\theta;$
    $i :=$ thread(i);
while $i \neq r_{tr}$

the following, we will first present an example that features four main procedures for updating indices (Algorithms 6.18, 6.19, 6.20, and 6.21). Then, we will integrate these four procedures into one algorithm (Algorithm 6.22) that outlines the update of tree

indices during a pivot operation for all types of bicycles and for all types of positions of the leaving arc. Algorithm 6.22 combines the four main update procedures for each case. Note that in some cases not all of the four update procedures are necessary and that in some cases some update procedures are applied more than once.

To illustrate the update of the tree indices, we provide an example that refers to the bicycle given in Figure 6.7(a) on page 168, which originated from only one old 1-tree tr. This bicycle features two cycles separated by a path. The bicycle is depicted again in Figure 6.8(a) on the current page supplemented with arc numbers. The tree indices are reported in Part 1 of Table 6.3, which is headed with $r_{tr} = 0$. Let us assume that the leaving arc $\langle p, q\rangle$ is the arc $\langle 1, 2\rangle$. Under this assumption, the leaving arc is part of the old cycle. Hence, only one new 1-tree tr' will emerge. The old extra arc $a_{tr}^{\text{extra}} = 8$ will become an ordinary arc of the new 1-tree. The cycle of the new 1-tree tr' will be formed by the nodes 5, 6, and 7, which form the new cycle in the bicycle, and we choose the entering arc $\langle k, l\rangle$ as the new extra arc, i.e., $a_{tr'}^{\text{extra}} := 9$ (cf. Figure 6.8(b)). Whether node $k = 6$, as shown in Figure 6.8(b), or node $l = 7$ will become the new root node will be decided later. In the subsequent paragraphs, we will work through this example for a tree update step by step.



(a) Old 1-tree tr with entering arc **(b)** New 1-tree tr'

Figure 6.8: Example for a tree update (arc $\langle 1, 2\rangle$ is the leaving arc)

The leaving arc causes the old spanning tree $G^{tr} = (N^{tr}, A^{tr})$ to break into two parts.[9] Let tr_1 denote the part or subgraph that contains the root node of the old 1-tree tr. In the example, the part tr_1 contains the root node 0, node 1, and arc $\langle 0, 1\rangle$. The second part, called tr_2, contains the nodes 2 to 7 and the arcs 3 to 7. In the following four steps, we construct the new 1-tree for which both parts are rejoined via the old extra arc $a_{tr}^{\text{extra}} = 8$.

In a first step, we have to identify the two parts tr_1 and tr_2. From the fact that node $q = 2$ is farther away from the root node 0 of the old 1-tree than node $p = 1$ (depth(2) > depth(1)), we can conclude that node $p = 1$ belongs to the first part tr_1,

[9]Note that A^{tr} includes neither the extra arc a_{tr}^{extra} nor the entering arc $\langle k, l\rangle$.

Table 6.3: Tree indices in the course of the update of the 1-tree illustrated in Figure 6.8: Part 1: indices for the old 1-tree, Part 2 and 3: indices for the intermediate trees tr_1 and tr_2, Part 4: indices for the new 1-tree tr'. The values that changed from step to step are printed in boldface.

i	0	1	2	3	4	5	6	7
	Part 1: $r_{tr} = 0$							
pred(i)	−1	0	1	2	3	4	5	5
depth(i)	0	1	2	3	4	5	6	6
thread(i)	1	2	3	4	5	6	7	0
revThread(i)	7	0	1	2	3	4	5	6
final(i)	7	7	7	7	7	7	6	7
SubtreeNodes(i)	8	7	6	5	4	3	1	1
ArcAndOrient(i)	8	1	2	3	4	5	6	7
Root(i)	0	0	0	0	0	0	0	0
	Part 2: $r_{tr_1} = 0 \quad r_{tr_2} = 2$							
pred(i)	−1	0	**−1**	2	3	4	5	5
depth(i)	0	1	**0**	**1**	**2**	**3**	**4**	**4**
thread(i)	1	**0**	3	4	5	6	7	**2**
revThread(i)	**1**	0	**7**	2	3	4	5	6
final(i)	**1**	**1**	7	7	7	7	6	7
SubtreeNodes(i)	**2**	**1**	6	5	4	3	1	1
ArcAndOrient(i)	8	1	2	3	4	5	6	7
Root(i)	0	0	**2**	**2**	**2**	**2**	**2**	**2**
	Part 3: $r_{tr_1} = 0 \quad r_{tr_2} = 6$							
pred(i)	−1	0	**3**	**4**	**5**	**6**	**−1**	5
depth(i)	0	1	**4**	**3**	2	**1**	**0**	**2**
thread(i)	1	0	**6**	**2**	**3**	**7**	**5**	**4**
revThread(i)	1	0	**3**	**4**	**7**	**6**	**2**	**5**
final(i)	1	1	**2**	**2**	**2**	**2**	**2**	7
SubtreeNodes(i)	2	1	**1**	**2**	**3**	**5**	**6**	1
ArcAndOrient(i)	8	1	**−3**	**−4**	**−5**	**−6**	6	7
Root(i)	0	0	**6**	**6**	**6**	**6**	**6**	**6**
	Part 4: $r_{tr'} = 6$							
pred(i)	**3**	0	3	4	5	6	−1	5
depth(i)	**4**	**5**	4	3	2	1	0	2
thread(i)	1	**6**	**0**	2	3	7	5	4
revThread(i)	**2**	0	3	4	7	6	**1**	5
final(i)	1	1	2	**1**	**1**	**1**	**1**	7
SubtreeNodes(i)	2	1	1	**4**	**5**	**7**	**8**	1
ArcAndOrient(i)	**−8**	1	−3	−4	−5	−6	**9**	7
Root(i)	**6**	**6**	6	6	6	6	6	6

which contains the root node, and is placed just before the breakage. This node $p = 1$,

the node belonging to the first part tr_1, is termed n_1. Furthermore, we can conclude that $tr_2 = tr(q) = tr(2)$, i.e., that the second part is given by the subtree $tr(2)$ including node 2 and its successors. We denote the node $q = 2$, the node which is placed immediately behind the breakage, by n_2.

In a second step, we update the tree indices of both parts tr_1 and tr_2 such that each part is correctly described by its indices. To update the indices of the tree tr_1 we apply Algorithm 6.18, which is the first of the four main update procedures. This algorithm updates final nodes, thread indices, reverse thread indices and numbers of subtree nodes of the nodes belonging to tr_1, whereas predecessors and depth indices do not change for all nodes $i \in N^{tr_1}$. Our implementation follows the outline of Bazaraa et al. (2010, p. 487) and uses their notation for identifiers that have not already been introduced in this thesis.[10]

From line 4 to 20 of Algorithm 6.18, the index final(i), $i \in N^{tr_1}$, is updated. The parameter σ is set to the final node of n_1 if the final node of n_1 belongs to tr_1. Otherwise, σ is set to the reverse thread of node n_2, where revThread(n_2) $\in N^{tr_1}$. For our example, the latter case holds and we set $\sigma :=$ revThread($n_2 = 2$) $= 1$. The node σ becomes the new final node of each node on the path from n_1 in direction of the root node up to the last node i with final(i) $\in tr_2$. This last node i is identified with the help of the parameter γ. If $\gamma = -1$, as in our example, we set the final node of every node on the path including the root node to σ. Otherwise, only the final nodes of the nodes up to node γ but not including node γ are set to σ.

From line 21 to 23 of Algorithm 6.18, one thread index and the corresponding reverse thread index are updated: ϕ denotes the node from which the thread enters the tree tr_2, while θ identifies the node of re-entrance of the thread into tr_1. For our example, $\phi = 1$ and $\theta = 0$. The updated thread directly connects the nodes ϕ and θ. Finally, Algorithm 6.18 decreases the number of subtree nodes for each node $i \in N^{tr_1}$ that lies on the path from n_1 to r_{tr_1} by $|N^{tr_2}|$. Algorithm 6.18 runs in $O\left(|N^{tr_1}| + |N^{tr_2}|\right)$ time.

To update the indices of the second tree tr_2, we apply Algorithm 6.19, which is the second of the four main update procedures. This algorithm makes node n_2 the root node of tr_2 and starts with updating the thread index of the final node of node n_2, which still points to a node of tr_1. Thus, for our example in Figure 6.8, the thread index of node 7 is set to 2. After having updated the corresponding reverse thread index, the depth values of n_2 and its successors are corrected. Additionally, Algorithm 6.19 sets the root node index for each node that belongs to tr_2 to node n_2. Algorithm 6.19 requires $O\left(|N^{tr_2}|\right)$ time.

For our example, the updated indices of tr_1 and tr_2 are stated in Part 2 of Table 6.3 on the facing page.[11] The values that changed are printed in boldface in this table.

In a third step on our way to the new 1-tree tr', we make either node k or node l the new root node and reroot the corresponding tree to this node. In the case of one old 1-tree with non-overlapping cycles where the leaving arc belongs to the old cycle, node k and l belong either to tree tr_1 or to tree tr_2. The question to which tree k and l belong is answered by the indices Root(k) and Root(l), because either Root(k) = Root(l) = r_{tr}

[10] With respect to identifiers that have already been introduced, our notation differs from the notation of Bazaraa et al. (2010).

[11] Unlike Algorithm 6.19, we set the predecessor index of node 2 to -1 in Table 6.3 on the preceding page to indicate the two separate parts tr_1 and tr_2.

Algorithm 6.18 Algorithm to update the indices of tree tr_1 containing the old root node r_{tr}

Input: Root node r_{tr} of the old 1-tree tr, nodes n_1 and n_2
Output: Updated tree indices for the nodes of tree tr_1

$\gamma := \text{pred}\big(\text{thread}(\text{final}(n_2))\big)$;
$\sigma := \text{final}(n_1)$
for $i := n_2$, $j := 0$; $j < \text{SubtreeNodes}(n_2)$; $++j$ **do**
    if $i = \sigma$ **then**
        $\sigma := \text{revThread}(n_2)$;
        break; // Abort the for-loop
    $i := \text{thread}(i)$;
$i := n_1$;
if $\gamma = -1$ **then**
    while $i \neq r_{tr}$ **do**
        $\text{final}(i) := \sigma$;
        $i := \text{pred}(i)$;
    $\text{final}(r_{tr}) := \sigma$;
else
    while $i \neq \gamma$ **do**
        $\text{final}(i) := \sigma$;
        $i := \text{pred}(i)$;
$\phi := \text{revThread}(n_2)$;
$\theta := \text{thread}(\text{final}(n_2))$;
$\text{thread}(\phi) := \theta, \quad \text{revThread}(\theta) := \phi$;
$i := n_1$;
while $i \neq r_{tr}$ **do**
    $\text{SubtreeNodes}(i) := \text{SubtreeNodes}(i) - \text{SubtreeNodes}(n_2)$;
    $i := \text{pred}(i)$;
$\text{SubtreeNodes}(r_{tr}) := \text{SubtreeNodes}(r_{tr}) - \text{SubtreeNodes}(n_2)$;

or $\text{Root}(k) = \text{Root}(l) = n_2$ holds. The latter is true for our example in Figure 6.8, i.e., k and l belong to tree tr_2. To decide whether node k or node l becomes the new root node of tree tr_2, the length of the so called stem is considered. The stem is the path between the old and the new root node. The shorter the stem, the less effort is necessary to reroot the tree (cf. Bazaraa et al., 2010, pp. 487–488). In our example, the two stems are the paths from node $n_2 = 2$ to node $k = 6$ and to node $l = 7$, respectively. Both stems have equal length ($\text{depth}(6) = \text{depth}(7)$). To break the tie, we select the from-node $k = 6$ of the entering arc to become the new root node of tree tr_2. Now, we reroot tree tr_2 from its old root node $n_2 = 2$ to its new root node $r_{tr_2}^{\text{new}}$=6.

For rerooting a tree and updating the tree indices accordingly, Algorithm 6.20 is applied, which is the third of the four main update procedures. Algorithm 6.20 describes how indices of a tree tr must be updated when the tree tr is rerooted from his old root node r_{tr}^{old} to a new root node r_{tr}^{new}. We derived this algorithm from the outline in Bazaraa

Algorithm 6.19 Algorithm to update the indices of tree tr_2, which will be rooted at node n_2

1: **Input:** Root node n_2 of the tree tr_2
2: **Output:** Updated tree indices for the nodes of tree tr_2
3: ---
4: $\text{thread}(\text{final}(n_2)) := n_2, \quad \text{revThread}(n_2) := \text{final}(n_2);$
5: $\Delta^{\text{depth}} := \text{depth}(n_2);$
6: $\text{depth}(n_2) := 0;$
7: $\text{Root}(n_2) := n_2;$
8: $i := \text{thread}(n_2);$
9: **while** $i \neq n_2$ **do**
10: $\quad \text{depth}(i) := \text{depth}(i) - \Delta^{\text{depth}};$
11: $\quad \text{Root}(i) := n_2;$
12: $\quad i := \text{thread}(i);$

et al. (2010, pp. 487–488). The algorithm requires the old root node r_{tr}^{old} and the new root node r_{tr}^{new} as input.

For the following description of Algorithm 6.20, we refer to the tree sketched in Figure 6.9, where the stem is given by the node sequence r_{tr}^{new}–s_1–s_2–s_3–r_{tr}^{old} (Figure 6.9 is adopted from Figure 9.19(c) on page 486 in Bazaraa et al. (2010), which illustrates the underlying tree structure along the stem).

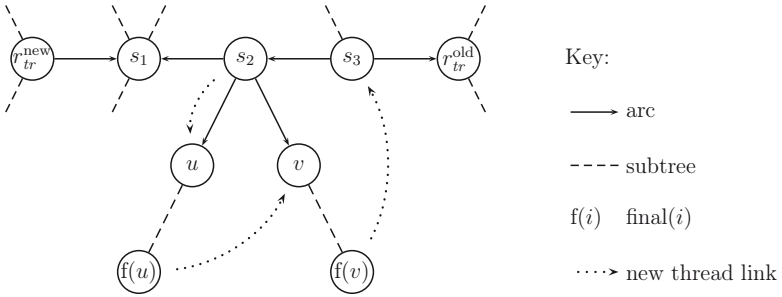


Figure 6.9: Sketch of the tree structure along the stem between the new root node r_{tr}^{new} and the old root node r_{tr}^{old} (adopted from Figure 9.19(c) on page 486 in Bazaraa et al. (2010))

At the outset, Algorithm 6.20 saves a copy of the vectors thread(i) and SubtreeNodes(i), $i \in N^{tr}$, because at some points within the algorithm old values of these indices are required when they have already been updated. Next, the algorithm sets the thread index of the final node of r_{tr}^{new} to the predecessor of r_{tr}^{new}, which is the node that is adjacent to r_{tr}^{new} and that is on the stem. This node is node s_1. To update the depth indices, we use the distance of the new root node from the old root node. The distance is stored in the old value of the depth index of the new root node. In line 8

of Algorithm 6.20, we initialize the variable Δ^{depth} with this distance. Then, the depth indices of the new root node and its successors are updated, i.e., decreased by Δ^{depth}.

Algorithm 6.20 Algorithm to reroot a tree tr

Input: Old root node r_{tr}^{old} of the tree tr, new root node r_{tr}^{new}
Output: Updated tree indices for the nodes of tree tr

---

for all $i \in N^{tr}$ **do**
    oldThread(i) := thread(i);
    oldSubtreeNodes(i) := SubtreeNodes(i);
thread(final(r_{tr}^{new})) := pred(r_{tr}^{new}), revThread(pred(r_{tr}^{new})) := final(r_{tr}^{new});
Δ^{depth} := depth(r_{tr}^{new});
for $i := r_{tr}^{\text{new}}$, $j := 0$; $j <$ OldSubtreeNodes(r_{tr}^{new}); $++j$ **do**
    depth(i) := depth(i) $- \Delta^{\text{depth}}$;
    i := thread(i);
$j := r_{tr}^{\text{new}}$;
newDepth := 0;
do // First traversal of the stem
    i := pred(j);
    newDepth := newDepth + 1;
    depth(i) := newDepth;
    $\Delta^{\text{depth}} := \Delta^{\text{depth}} - 2$;
    SubtreeNodes(i) := 1; // Initialization of SubtreeNodes(i)
    RelevSubtreeNodes := OldSubtreeNodes(i);
    if RelevSubtreeNodes > OldSubtreeNodes(j) + 1 **then** // If node i has immediate successors that do not lie on the stem
        u := oldThread(i);
        if $u \neq j$ **then** // If node u does not lie on the stem
            thread(i) := u, revThread(u) := i;
        else
            u := oldThread(final(u));
            thread(i) := u, revThread(u) := i;
        for $c := u$, $d := 0$; $d <$ OldSubtreeNodes(u); $++d$ **do**
            depth(c) := depth(c) $- \Delta^{\text{depth}}$;
            c := thread(c);
        SubtreeNodes(i) := SubtreeNodes(i) + OldSubtreeNodes(u);
        while RelevSubtreeNodes > OldSubtreeNodes(j) + 1 **do** // While not all immediate successors of node i have been considered
            if $u \neq j$ **then** // If node u does not lie on the stem
                RelevSubtreeNodes := RelevSubtreeNodes − OldSubtreeNodes(u);
                if RelevSubtreeNodes > OldSubtreeNodes(j) + 1 **then**
                    v := oldThread(final(u));
                    if $v \neq j$ **then** // If node v does not lie on the stem
                        thread(final(u)) := v, revThread(v) := final(u);
                    else

```
40:                         v := oldThread(final(v));
41:                         thread(final(u)) := v,    revThread(v) := final(u);
42:                     for c := v, d := 0; d < OldSubtreeNodes(v); ++d do
43:                         depth(c) := depth(c) − Δ^depth;
44:                         c := thread(c);
45:                     SubtreeNodes(i) := SubtreeNodes(i) + OvdSubtreeNodes(v);
46:                 else
47:                     if i ≠ r_tr^old then
48:                         thread(final(u)) := pred(i),    revThread(pred(i)) := final(u);
49:                     else
50:                         thread(final(u)) := r_tr^new,    revThread(r_tr^new) := final(u);
51:                         NewFinal := final(u);
52:                     break;                          // Abort the while-loop started in line 32
53:             u := v;
54:     else                                            // Corresponds to the "if" in line 21
55:         if i ≠ r_tr^old then
56:             thread(i) := pred(i),    revThread(pred(i)) := i;
57:         else
58:             thread(r_tr^old) := r_tr^new,    revThread(r_tr^new) := r_tr^old;
59:             NewFinal := r_tr^old;
60:     j := i;
61: while j ≠ r_tr^old                                  // End of do-loop started in line 14
62: j := r_tr^new;
63: final(r_tr^new) := NewFinal;
64: i := pred(j);
65: k := i;
66: NewArc := ArcAndOrient(j);
67: do                                                  // Second traversal of the stem
68:     OldArc := ArcAndOrient(i);
69:     ArcAndOrient(i) := −NewArc;
70:     NewArc := OldArc;
71:     final(i) := NewFinal;
72:     i := pred(i);
73:     pred(k) := j;
74:     j := k;
75:     k := i;
76: while j ≠ r_tr^old
77: pred(r_tr^new) := −1;
78: j := r_tr^old;
79: do                                                  // Third traversal of the stem
80:     i := pred(j);
81:     SubtreeNodes(i) := SubtreeNodes(i) + SubtreeNodes(j);
82:     j := i;
83: while j ≠ r_tr^new
```

In the remainder of Algorithm 6.20, the stem is traversed three times to update indices. The first traversal runs from the new to the old root node. In this traversal, the indices depth(i), thread(i), and revThread(i) are updated, and the update of the index SubtreeNodes(i) of those nodes i that lie on the stem is begun.

Let us describe this preliminary update of the number of subtree nodes during the first traversal more precisely. For each node i on the stem except for r_{tr}^{new}, we update depth(i) and decrease Δ^{depth} by 2 reflecting that approaching r_{tr}^{old} means moving away from r_{tr}^{new} during traversal of the stem. Additionally, we initialize for each node i on the stem its number of subtree nodes with 1 and we check if node i has immediate successors that do not lie on the stem, which we term *immediate non-stem successors*. For example, consider node s_2 on the stem, which has nodes u and v as immediate non-stem successors. We can conclude that node s_2 has at least one immediate non-stem successor if its subtree nodes do comprise more than node s_2 and the subtree nodes of s_1. Hence, we compare SubtreeNodes(s_2) and SubtreeNodes(s_1) to find out whether node s_2 has at least one immediate non-stem successor (cf. line 21 of Algoritm 6.20).

If we find out that s_2 has at least one immediate non-stem successor, we select an immediate successor and denote this successor by u (line 22). If node u happens to lie on the stem, i.e., if $u = s_1$, we just select another immediate successor and denote it by u. This successor cannot lie on the stem with certainty. We link s_2 with u by the thread index of s_2. For the successors of u, we update their depth values using Δ^{depth}. The current number of subtree nodes of s_2 is increased by the number of subtree nodes of u. If there is another immediate non-stem successor of s_2, say node v, we link the final node of u with v via the thread index of final(u), then we update the depth values of the successors of node v and increase SubtreeNodes(s_2) by SubtreeNodes(v). In this way, we treat all immediate non-stem successors of s_2. For the last immediate non-stem successor of node s_2, say node u, we link the final node of u via the thread index with s_3 (line 48).

If we find out that s_2 does not have any immediate non-stem successor, we directly link s_2 with s_3 via the thread index of s_2 (line 56).

When the first traversal reaches the old root node r_{tr}^{old}, the same procedure as for s_2 applies. However, we additionally record the new final node of r_{tr}^{new} as NewFinal, either in line 51 if r_{tr}^{old} has at least one immediate non-stem successor or in line 59 otherwise.

The second traversal of the stem runs again from the new to the old root node. During this traversal, we update the indices final(i), pred(i), and ArcAndOrient(i) of those nodes i that lie on the stem. The index final(i) of these nodes i is set to NewFinal. For pred(i) and ArcAndOrient(i), the update corresponds to a reversion of the indices. For instance, for node s_2 node s_1 becomes the new predecessor replacing node s_3. The arc $\langle s_2, s_1 \rangle$, which was an outbound arc with respect to r_{tr}^{old}, becomes an inbound arc with respect to r_{tr}^{new}.

Finally, the stem is traversed in opposite direction, from r_{tr}^{old} to r_{tr}^{new}, to finish the update of the index SubtreeNodes(i) of those nodes i that lie on the stem. For node s_2, for example, the number of subtree nodes is increased by SubtreeNodes(s_3) and SubtreeNodes(r_{tr}^{old}), as recorded after the first traversal.

Since during each traversal each node of the tree tr is visited at most once, Algorithm 6.20 runs in $O(|N^{tr}|)$ time.

Two examples illustrate the outcome of Algorithm 6.20. For our example shown in Figure 6.8 on page 171, the updated indices of tr_2 are stated in Part 3 of Table 6.3 on

page 172. Table 6.2 on page 164 gives a further example, this time for a larger tree, which is depicted in Figure 6.5 on page 163. This larger tree is rerooted from node 0 to node 8.

In a fourth step, which is the final step of constructing the new 1-tree, the two parts tr_1 and tr_2 are rejoined again to form the new 1-tree tr'. The arc that links both parts is the old extra arc $\langle \alpha, \beta \rangle$, which will become an ordinary arc of the new 1-tree tr'. In our example in Figure 6.8 on page 171, the old extra arc is arc $\langle 0, 3 \rangle$ and we graft tree tr_1 onto node $\beta = 3$, which belongs to tree tr_2. Together, both trees form the new 1-tree tr' with root node $r_{tr'} = 6$. In general, either node α or node β belongs to the part $tr(r_{tr'})$ that contains the root node of the new 1-tree. Let n^{onto} denote the node of the tree $tr(r'_{tr})$ onto which the other part tr^{toGraft} will be grafted:

$$n^{\text{onto}} := \begin{cases} \alpha & \text{if } \alpha \in N^{tr(r_{tr'})} \\ \beta & \text{otherwise} \end{cases}$$

The part tr^{toGraft} that must be grafted onto node n^{onto} contains either node β or α. Let n^{toGraft} denote the node that will be grafted together with the tree tr^{toGraft} onto node n^{onto}:

$$n^{\text{toGraft}} := \begin{cases} \beta & \text{if } n^{\text{onto}} = \alpha \\ \alpha & \text{otherwise} \end{cases}$$

For our example, we have $n^{\text{onto}} := 3$ and $n^{\text{toGraft}} := 0$. Before we can graft the part tr^{toGraft} that contains node n^{toGraft} onto the part tr^{onto} that contains n^{onto}, we have to reroot the part tr^{toGraft} to node n^{toGraft} if it is not already rooted at this node. In our example, tree $tr^{\text{toGraft}} = tr_1$ is already rooted at node 0, what renders rerooting unnecessary.

For the grafting process, we apply Algorithm 6.21, which is the final of the four main update procedures. This algorithm follows the description of Bazaraa et al. (2010, pp. 486–487). Since the tree tr^{toGraft} is joined to the node n^{onto}, we update the depth indices of the nodes $i \in N^{tr^{\text{toGraft}}}$ based on the depth index of n^{onto}. $\text{Root}(i)$, $i \in N^{tr^{\text{toGraft}}}$, saves the root node of the tree tr^{onto} as the root node for the joined 1-tree tr'. The node n^{toGraft} is considered the last immediate successor of n^{onto}. Hence, we link the old final node of n^{onto}, node x, via the thread index with n^{toGraft} (cf. line 15 of Algorithm 6.21). The final node of n^{toGraft}, node z, becomes the new final node of n^{onto} and must therefore be linked with the thread node y of the old final node of n^{onto} (line 16). Next, we update the number of subtree nodes for the nodes on the path from n^{onto} to the root node of the tree tr^{onto}. At the same time, the index $\text{final}(i)$ is updated, either for all nodes on this path if node x was the old final node of the root node (case $\gamma = 1$) or only for those nodes i on this path that had x as their final node (case $\gamma \neq 1$). Algorithm 6.21 requires $O\left(\left|N^{tr^{\text{toGraft}}}\right| + \left|N^{tr^{\text{onto}}}\right|\right)$ time.

To complete the grafting operation, we set $\text{pred}(n^{\text{toGraft}}) := n^{\text{onto}}$, adjust $\text{ArcAndOrient}(n^{\text{toGraft}})$, and update information about the extra arc of the new 1-tree, e.g., the index $\text{ArcAndOrient}(r_{tr'})$ of the new root node. For our example, the resulting indices of this final step are stated in Part 4 of Table 6.3 on page 172. The corresponding new 1-tree tr' is shown in Figure 6.8(b) on page 171.

The way of updating the indices during a pivot operation depends on the bicycle that has been induced by the entering arc $\langle k, l \rangle$ and on the position of the leaving arc $\langle p, q \rangle$. For all possible cases of bicycle types and of positions of leaving arcs, the four main

Algorithm 6.21 Algorithm to graft a tree tr^{toGraft} onto node n^{onto} of another tree tr^{onto}

```
1: Input: Root node n^toGraft of the tree tr^toGraft, node n^onto and root node r_{tr^onto} of the
   other tree tr^onto
2: Output: Tree indices for the joined tree
3: ---------------------------------------------------------------------------
4: depth(n^toGraft) := depth(n^onto) + 1;
5: Root(n^toGraft) := r_{tr^onto};
6: i := thread(n^toGraft);
7: while i ≠ n^toGraft do
8:     depth(i) := depth(pred(i)) + 1;
9:     Root(i) := r_{tr^onto};
10:    i := thread(i);
11: x := final(n^onto);
12: γ := pred(thread(x));
13: y := thread(x);
14: z := final(n^toGraft);
15: thread(x) := n^toGraft,    revThread(n^toGraft) := x;
16: thread(z) := y,    revThread(y) := z;
17: i := n^onto;
18: if γ = −1 then
19:     do
20:         final(i) := z;
21:         SubtreeNodes(i) := SubtreeNodes(i) + SubtreeNodes(n^toGraft);
22:         i := pred(i);
23:     while i ≠ −1
24: else
25:     while i ≠ γ do
26:         final(i) := z;
27:         SubtreeNodes(i) := SubtreeNodes(i) + SubtreeNodes(n^toGraft);
28:         i := pred(i);
29:     do
30:         SubtreeNodes(i) := SubtreeNodes(i) + SubtreeNodes(n^toGraft);
31:         i := pred(i);
32:     while i ≠ −1
```

update procedures, which are Algorithms 6.18, 6.19, 6.20, and 6.21, must be composed in a special way for the update of the indices. Algorithm 6.22 outlines how the four main update procedures must be composed in each case. Note that in some cases not all of the four update procedures are necessary and that in some cases some update procedures are applied more than once.

To keep Algorithm 6.22 clear, we outsourced pseudo code to Algorithms 6.23, 6.24, and 6.25, and we abbreviated the four main update procedures. A call of Algorithm 6.18 is denoted by **UpdateTreeWithRoot**(r_{tr}, n_1, n_2), Algorithm 6.19 is invoked by **UpdateSubtree**(n_2), Algorithm 6.20 is invoked by **RerootTree**(r_{tr}^{old}, r_{tr}^{new}), and a call of Algorithm 6.21 is denoted by **GraftTree**(n^{toGraft}, n^{onto}, $r_{tr^{\text{onto}}}$). Furthermore, we

omitted statements for updates of the indices onOldCycle(i), onNewCycle(i), onPath(i), ArcAndOrient(i) and Root(i) for the sake of readability. The implemented version of Algorithm 6.22 features a set that contains the indices of the root nodes of all 1-trees, which form the good augmented forest. In our presentation, we omitted statements that are required to update this set of 1-tree root nodes, again for the sake of readability.

The complexity of Algorithm 6.22 is a linear combination of the complexities of the Algorithms 6.18, 6.19, 6.20, and 6.21. Let the bicycle that is formed by the entering arc be denoted by *bicycle* and let $N^{bicycle}$ be the set of nodes that belong to this bicycle. Then, Algorithm 6.22 runs in $O\left(\left|N^{bicycle}\right|\right)$ time.

After having updated the tree indices, we complete the pivot operation by the following operations. We call Algorithm 6.16 for the new 1-tree(s) to compute the node potentials of the corresponding nodes. We remove the entering arc $\langle k, l\rangle$ from the candidate list and from the set of nonbasic arcs that emanate from node k. In return, the leaving arc $\langle p, q\rangle$ is added to the set of nonbasic arcs that emanate from node p. Finally, we update the candidate list by removing those arcs that do not violate their optimality condition anymore.

Pivot operations are repeated until an optimal solution has been reached. An optimal solution is reached when the reconstruction of the candidate list results in an empty list.

Having outlined the generalized network simplex method, we will succinctly describe how it can be applied within our heuristic DROP. Assume that we reformulated a remaining linear program $\mathrm{LP}_t^{\mathrm{rem}}$ as a generalized minimum cost flow problem. For each pair (k, p) that has already been dropped in the course of DROP, the costs of the corresponding arc between stage 1 and 2 in the network that is associated with $\mathrm{LP}_t^{\mathrm{rem}}$ were set to a prohibitively high value. The flow on this arc between stage 1 and 2 represents the time which worker k spends for project p. Due to the high costs of this arc, in an optimal solution there will be no flow on this arc. This zero flow is desired, because worker k is no longer assigned to project p. Nevertheless, we must check if an optimal solution to the generalized minimum cost flow problem is feasible for the corresponding remaining linear program $\mathrm{LP}_t^{\mathrm{rem}}$. The solution is only feasible for $\mathrm{LP}_t^{\mathrm{rem}}$ if the flow on the backward arc from the terminal node to the source node is equal to the total demand of projects and departments, i.e., if the arcs between stage 4 and the terminal node are saturated.

6.2.3.4 Discussion of the implementation

In this subsection, we briefly discuss our implementation of the generalized network simplex method for the heuristic DROP. We consider potential problems such as cycling, highlight efficient strategies within our implementation, and indicate possible improvements.

As with the standard simplex method, two problems can arise when the generalized network simplex method is applied, namely, *cycling* and *stalling*. Cycling means, in the context of the generalized network simplex method, that pivot operations step from one good augmented forest to the next, but the forest of each step describes the same basis solution, and finally a forest is reached that has been encountered before. From this forest, the cycle starts anew and the generalized network simplex method does not terminate, but executes degenerate pivot operations infinitely.

Cycling can be prevented by special rules for selecting the entering and the leaving

Algorithm 6.22 Algorithm to update tree indices during a pivot operation

1: **Input:** From-node k and to-node l of the entering arc $\langle k, l\rangle$, from-node p and to-node q of the leaving arc $\langle p, q\rangle$
2: **Output:** Updated tree indices for the nodes of the bicycle induced by arc $\langle k, l\rangle$
3: ---
4: **if** $a_{kl} = a_{pq}$ **then** // If $\langle k, l\rangle = \langle p, q\rangle$
5: **goto** *End* in line 38;
6: FirstOldOneTree := Root(k), SecondOldOneTree := Root(l);
7: boolTwoOldOneTrees := **false**, boolOverlappingCycles := **false**;
8: **if** FirstOldOneTree = SecondOldOneTree **then** // If k and l belong to the same 1-tree
9: Update the index onNewCycle(i), $i \in N$;
10: **if** the old cycle of the 1-tree FirstOldOneTree and the new cycle overlap **then**
11: boolOverlappingCycles := **true**;
12: **else** // There is a path between the old and the new cycle
13: Update the index onPath(i), $i \in N$;
14: **else**
15: boolTwoOldOneTrees := **true**;
16: Update the index onPath(i), $i \in N$;
17: **if** boolTwoOldOneTrees := **false then** // If k and l belong to the same 1-tree
18: **if** $a_{pq} = a^{\text{extra}}_{\text{FirstOldOneTree}}$ **then** // If $\langle p, q\rangle$ is the extra arc of the first old 1-tree
19: Make arc $\langle k, l\rangle$ the extra arc of the emerging new 1-tree;
20: **if** $k \neq$ Root(FirstOldOneTree) **and** $l \neq$ Root(FirstOldOneTree) **then**
21: **RerootTree**(Root(FirstOldOneTree), k); // Or reroot to l if faster
22: **goto** *End* in line 38;
23: **else**
24: **if** boolOverlappingCycles := **true then** // If old and new cycle overlap, only one new 1-tree will emerge
25: **Algorithm 6.23**;
26: **goto** *End* in line 38;
27: **else** // If the old and the new cycle do not overlap (corresponds to the "if" in line 24)
28: **Algorithm 6.24**;
29: **goto** *End* in line 38;
30: **else** // If k and l belong to different 1-trees
31: **if** Root(p) = FirstOldOneTree **then** // If $\langle p, q\rangle$ belongs to the first old 1-tree
32: boolLeavingArcInFirstOldOneTree := **true**;
33: TreeOfLeavingArc := FirstOldOneTree;
34: **else** // If $\langle p, q\rangle$ belongs to the second old 1-tree
35: boolLeavingArcInFirstOldOneTree := **false**;
36: TreeOfLeavingArc := SecondOldOneTree;
37: **Algorithm 6.25**;
38: *End*:

Algorithm 6.23 Subprocedure of Algorithm 6.22: One old 1-tree, overlapping cycles

if depth(p) < depth(q) **then**
  $n_1 := p, \quad n_2 := q;$
else
  $n_1 := q, \quad n_2 := p;$
UpdateTreeWithRoot(Root(FirstOldOneTree), n_1, n_2);
UpdateSubtree(n_2);
if onNewCycle(p) = **false or** onNewCycle(q) = **false then** // If $\langle p, q\rangle$ belongs to the old cycle only, make $\langle \alpha, \beta\rangle$ an ordinary arc and $\langle k, l\rangle$ the new extra arc
  if α = Root(FirstOldOneTree) **then** // If α belongs to tr_1
    $n^{\text{toRerootTo}} := \beta;$ // Reroot $tr_2 = tr(n_2)$ to β
  else // If β belongs to tr_1
    $n^{\text{toRerootTo}} := \alpha;$ // Reroot $tr_2 = tr(n_2)$ to α
  RerootTree(n_2, $n^{\text{toRerootTo}}$);
  GraftTree($n^{\text{toRerootTo}}$, Root(FirstOldOneTree), Root(FirstOldOneTree));
  RerootTree(Root(FirstOldOneTree), k); // Or reroot to l if faster
else // If $\langle p, q\rangle$ belongs to either the new cycle only or to both cycles, keep $\langle \alpha, \beta\rangle$ as extra arc and make $\langle k, l\rangle$ an ordinary arc
  if Root(k) = n_2 **then** // If k belongs to tr_2
    $n^{\text{toRerootTo}} := k, \quad n^{\text{onto}} := l;$
  else // If l belongs to tr_2
    $n^{\text{toRerootTo}} := l, \quad n^{\text{onto}} := k;$
  RerootTree(n_2, $n^{\text{toRerootTo}}$); // Reroot $tr_2 = tr(n_2)$ to $n^{\text{toRerootTo}}$
  GraftTree($n^{\text{toRerootTo}}$, n^{onto}, Root(FirstOldOneTree));

arc. An example for such a pivoting rule is Bland's smallest-subscript rule (cf. Chvátal, 1983, pp. 37–38). Another way of avoiding cycling is to restrict the forests that are visited to so called strongly convergent bases (cf. Elam et al., 1979). A strongly convergent basis is a good augmented forest that features a special topology. To facilitate this special topology, an artificial mirror arc must be established for each network arc. A mirror arc points into opposite direction of its corresponding ordinary arc. To maintain the special topology when a pivot operation is executed, a unique arc must be selected as the leaving arc.

Since cycling seldom appears in practical applications (cf. Chvátal, 1983, p. 33) and it did not occur for the instances that we tested, we did not take up the concept of strongly feasible bases in our implementation. Moreover, we did not apply Bland's pivoting rule, but applied a more promising rule with respect to average solution time.

Stalling, the second potential problem, terms a sequence of degenerate pivot operations where the length of this sequence is finite, but not bounded by a polynomial in the instance size. For the (non-generalized) minimum cost flow problem, Cunningham (1979) showed that stalling can occur even if cycling is prevented. He outlined anti-stalling rules that prevent stalling by choosing the entering arc according to certain rules. For the generalized minimum cost flow problem, no anti-stalling rules are known, but stalling is not supposed to be of relevance in practice (cf. Orlin, 2013).

Two strategies within our implementation save computational effort. The first strategy

Algorithm 6.24 Subprocedure of Algorithm 6.22: One old 1-tree, cycles do not overlap

```
1: boolLeavingArcBelongsToOldCycle := false;
2: if onOldCycle(p) = true and onOldCycle(q) = true then      // If ⟨p, q⟩ belongs to the old cycle
3:     boolLeavingArcBelongsToOldCycle := true;
4: if depth(p) < depth(q) then
5:     n_1 := p,    n_2 := q;
6: else
7:     n_1 := q,    n_2 := p;
8: UpdateTreeWithRoot(Root(FirstOldOneTree), n_1, n_2);
9: UpdateSubtree(n_2);
10: if boolLeavingArcBelongsToOldCycle = true then    // ⟨α, β⟩ becomes an ordinary arc, make ⟨k, l⟩ the new extra arc
11:     if Root(k) = Root(FirstOldOneTree) then              // If k and l belong to tr_1
12:         r^old := Root(FirstOldOneTree);                  // Reroot tr_1 to k or l
13:         r_{tr^toGraft} := n_2;                           // Graft tr_2 onto α or β
14:     else                                                 // If k and l belong to tr_2
15:         r^old := Root(FirstOldOneTree);                  // Reroot tr_2 to k or l
16:         r_{tr^toGraft} := Root(FirstOldOneTree);         // Graft tr_1 onto α or β
17:     RerootTree(r^old, k);                                // Or reroot to l if faster
18:     FirstNewOneTree := k;                                // Or FirstNewOneTree := l
19:     if Root(α) = r_{tr^toGraft} then                     // If α belongs to the tree tr^toGraft
20:         n^toRerootTo := α,    n^onto := β;
21:     else                                                 // If β belongs to the tree tr^toGraft
22:         n^toRerootTo := β,    n^onto := α;
23:     RerootTree(r_{tr^toGraft}, n^toRerootTo);
24:     GraftTree(n^toRerootTo, n^onto, FirstNewOneTree);
25: else
26:     if onNewCycle(p) = true and onNewCycle(q) = true then    // If ⟨p, q⟩ belongs to the new cycle, ⟨α, β⟩ is kept as extra arc, ⟨k, l⟩ becomes an ordinary arc
27:         if Root(k) = n_2 then                            // If k belongs to tr_2 and l to tr_1
28:             n^toRerootTo := k    n^onto := l;
29:         else                                             // If l belongs to tr_2 and k to tr_1
30:             n^toRerootTo := l    n^onto := k;
31:         RerootTree(n_2, n^toRerootTo);                   // Reroot tr_2
32:         GraftTree(n^toRerootTo, n^onto, Root(n^onto));   // Graft tr_2 onto tr_1
33:     else    // If ⟨p, q⟩ belongs to the path between the old and new cycle, two new 1-trees will emerge, ⟨α, β⟩ is kept as extra arc for the first new 1-tree (tr_1), ⟨k, l⟩ becomes the extra arc of the second new 1-tree (tr_2)
34:         RerootTree(n_2, k);                              // Or reroot to l if faster
```

applies when we traverse the bicycle to determine the leaving arc. If we encounter an arc whose flow cannot be changed, i.e., an arc that results in $\delta = 0$, this arc renders the

Algorithm 6.25 Subprocedure of Algorithm 6.22: Two old 1-trees

1: **if** onOldCycle(p) = **true and** onOldCycle(q) = **true then** // If $\langle p, q\rangle$ belongs to a cycle, only one new 1-tree will emerge
2: **if** $a_{pq} \neq a^{\text{extra}}_{\text{TreeOfLeavingArc}}$ **then** // If $\langle p, q\rangle$ is not the extra arc of the 1-tree TreeOfLeavingArc
3: **if** depth(p) < depth(q) **then**
4: $n_1 := p, \quad n_2 := q;$
5: **else**
6: $n_1 := q, \quad n_2 := p;$
7: **UpdateTreeWithRoot**(Root(TreeOfLeavingArc), n_1, n_2);
8: **UpdateSubtree**(n_2);
9: **if** ArcAndOrient(Root(TreeOfLeavingArc)) > 0 **then** // If the extra arc $a^{\text{extra}}_{\text{TreeOfLeavingArc}}$ ($\langle \alpha, \beta\rangle$) is outbound
10: $n^{\text{toRerootTo}} := \beta;$ // Reroot tr_2 to β
11: **else** // If the extra arc $a^{\text{extra}}_{\text{TreeOfLeavingArc}}$ ($\langle \alpha, \beta\rangle$) is inbound
12: $n^{\text{toRerootTo}} := \alpha;$ // Reroot tr_2 to α
13: **RerootTree**(n_2, $n^{\text{toRerootTo}}$);
14: **GraftTree**($n^{\text{toRerootTo}}$, Root(TreeOfLeavingArc), TreeOfLeavingArc); // Graft tr_2 onto tr_1 via $\langle \alpha, \beta\rangle$
15: **if** boolLeavingArcInFirstOldOneTree = **true then** // If $\langle p, q\rangle$ belongs to the first old 1-tree (associated with k)
16: TreeWithNewRoot := SecondOldOneTree;
17: $n^{\text{toRerootTo}} := k \quad n^{\text{onto}} := l;$
18: **else** // If $\langle p, q\rangle$ belongs to the second old 1-tree (associated with l)
19: TreeWithNewRoot := FirstOldOneTree;
20: $n^{\text{toRerootTo}} := l \quad n^{\text{onto}} := k;$
21: **RerootTree**(Root(TreeOfLeavingArc), $n^{\text{toRerootTo}}$);
22: **GraftTree**($n^{\text{toRerootTo}}$, n^{onto}, TreeWithNewRoot);
23: **else** // If $\langle p, q\rangle$ lies on a path between both cycles, two new 1-trees will emerge
24: **if** depth(p) < depth(q) **then**
25: $n_1 := p, \quad n_2 := q;$
26: **else**
27: $n_1 := q, \quad n_2 := p;$
28: **UpdateTreeWithRoot**(Root(TreeOfLeavingArc), n_1, n_2);
29: **UpdateSubtree**(n_2);
30: **if** boolLeavingArcInFirstOldOneTree = **true then** // If $\langle p, q\rangle$ belongs to the first old 1-tree (associated with k)
31: TreeWithNewRoot := SecondOldOneTree;
32: $n^{\text{toRerootTo}} := k \quad n^{\text{onto}} := l;$
33: **else** // If $\langle p, q\rangle$ belongs to the second old 1-tree (associated with l)
34: TreeWithNewRoot := FirstOldOneTree;
35: $n^{\text{toRerootTo}} := l \quad n^{\text{onto}} := k;$
36: **RerootTree**(n_2, $n^{\text{toRerootTo}}$);
37: **GraftTree**($n^{\text{toRerootTo}}$, n^{onto}, TreeWithNewRoot);

pivot operation degenerate.[12] We immediately make this arc the leaving arc and abort the traversal to save time (cf. Brown and McBride, 1984, p. 1508).

The second strategy is applied whenever a pair (k, p) was successfully dropped, i.e., when the corresponding remaining linear programs $\mathrm{LP}_t^{\mathrm{rem}}$, $t \in \mathcal{T}_p$, are found to be feasible. Then, we consider each network that corresponds to a period $t \in \mathcal{T}_p$ where $r_{pst} > 0$ holds for at least one skill $s \in \mathcal{S}_{kp}^{\mathrm{match}}$. In such a period t, worker k could contribute to project p if he was assigned to p. Hence, the network of period t contains an arc that runs from the node at stage 1 which represents worker k to the node at stage 2 that represents worker k and project p. For this arc and all its successors that run from stage 2 to stage 3 and from stage 3 to stage 4, we check if the respective arc is a nonbasic arc. If so, the arc must carry zero flow now. We delete this arc from the set of nonbasic arcs, because the arc cannot enter the basis anymore, as the pair (k, p) was dropped. The deletion of these nonbasic arcs saves time, as reduced cost calculations are saved when the network is reoptimized in later drop operations.

Example 6.11 To give an example for the second time-saving strategy, consider the network in Figure 6.4 on page 159 and assume that the pair (k_2, p_1) was successfully dropped. Then, we check whether arcs $\langle 2, 5\rangle$, $\langle 5, 10\rangle$, $\langle 5, 11\rangle$, $\langle 10, 13\rangle$, and $\langle 11, 14\rangle$ are nonbasic arcs. Those that are nonbasic arcs are removed from the set of nonbasic arcs. □

There are other possibilities for improving our approach, e.g., applying a polynomial-time algorithm to the generalized minimum cost flow problem or parallelizing computations. Although our implementation is straightforward and efficient from a practical point of view, its worst-case time complexity is exponential (cf. Ahuja et al., 1993, p. 590). There are combinatorial algorithms that exploit the underlying network structure and that run in polynomial time, e.g., the minimum ratio circuit-canceling algorithm of Wayne (2002). Furthermore, versions of the generalized network simplex method were outlined that exploit the enhancements offered by parallel computing (cf. Chang et al., 1988; Clark et al., 1992). For example, pivot operations that affect different 1-trees can be parallelized. In our case, we could apply parallelization also at a higher level. During the drop operation for a pair (k, p), the remaining linear programs $\mathrm{LP}_t^{\mathrm{rem}}$, $t \in \mathcal{T}_p$, can be solved in parallel. We did not try to exploit parallelization for any of our heuristics. Parallelization might be an avenue for future work.

6.2.3.5 Properties and modifications of DROP

In this subsection, we consider two properties of DROP. The first property is that the objective functions of the remaining linear programs are constant functions. This property is the starting point for a modification of DROP. The modification applies primarily to those variants of DROP that use the standard or the network model; it should not be applied to the variant that uses the generalized minimum cost flow model. The second property of DROP that we consider is the capability to find a feasible solution for any instance and the principal capability to find an optimal solution for any instance.

Be it for the standard or the network model: Fixing all binary variables $\boldsymbol{x}$ leads to a linear program with a constant objective function value, because the remaining variables $\boldsymbol{y}$ appear only within the constraint sets. If the linear program is decomposed into a set

[12] The identifier δ was introduced on page 165. How δ is computed was explained on page 169.

of remaining linear programs $\mathrm{LP}_t^{\mathrm{rem}}$, $t \in \mathcal{T}$, each of these LPs also features a constant objective function value. Hence, for each drop operation a set of feasibility problems must be solved.

Walter and Zimmermann (2010) proposed to replace the objective function of constant value in each linear program $\mathrm{LP}_t^{\mathrm{rem}}$ by a surrogate objective function that contains the corresponding variables y_{kpst}. Such a surrogate objective function has an impact on the values of the variables $\boldsymbol{y}$, which, in turn, may have two important impacts. First, the number of assignments (k, p) where worker k does not contribute to project p can change. Such a change may allow Algorithm 6.13 to delete more or other pairs (k, p) from the list $\mathcal{C}$, especially at the outset of the drop procedure when Algorithm 6.13 is called for the first time. Second, the selection probabilities of pairs (k, p) in the list $\mathcal{C}$ may change. Though, this impact on the selection probabilities can only occur if a dynamic unsuitability value is applied that takes the values of the variables y_{kpst} into account. Such a dynamic unsuitability value is defined in Equations (6.21) on page 153, for example.

As an example for a surrogate objective function for a remaining linear program $\mathrm{LP}_t^{\mathrm{rem}}$, $t \in \mathcal{T}$, consider the following objective function.

$$\text{Min.} \quad \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}_k^{\mathrm{suit}}(t)} \sum_{s \in \mathcal{S}_{kp}^{\mathrm{match}}} \frac{l_{ks}}{|\mathcal{S}_{kp}^{\mathrm{match}}|^2} \, y_{kpst} \tag{6.22}$$

Surrogate objective function (6.22) can be considered as a cost function which must be minimized. The costs arise for accomplishing the project workloads in period t. For each worker $k \in \mathcal{K}$, each project $p \in \mathcal{P}_k^{\mathrm{suit}}(t)$, and each skill $s \in \mathcal{S}_{kp}^{\mathrm{match}}$, we consider the workload $l_{ks} y_{kpst}$ that is accomplished by worker k for skill s of project p. The costs for this contribution of worker k to project p amount to $\frac{1}{|\mathcal{S}_{kp}^{\mathrm{match}}|^2}$. Hence, costs decline with an increasing number of matching skills between k and p.

The purpose of objective function (6.22) is to favor solutions for the corresponding remaining linear program in which each worker tends to contribute primarily to projects whose set of required skills matches his set of skills. The hope that these solutions lead to solutions of high quality for the underlying workforce assignment problem was the reason for designing this surrogate objective function.

Let us put forward two arguments why the solutions provided by objective function (6.22) should lead to relatively good solutions for the underlying problem in the course of DROP. First, an optimal solution for objective function (6.22) tends to enable Algorithm 6.13 to delete those pairs (k, p) from $\mathcal{C}$ where the number of matching skills between k and p is small, i.e., pairs (k, p) for which we expect $x_{kp} = 0$ in a high-quality solution for the underlying problem.

The second argument applies only if a dynamic unsuitability value us_{kp} is used that takes the values of the variables y_{kpst} into account. Hence, assume that we use unsuitability value us_{kp}^{C}, which is calculated based on Equations (6.21). Generally speaking, unsuitability value us_{kp}^{C} deems worker k the more unsuitable for a project p, the less time he spends for this project. In an optimal solution for objective function (6.22), worker k tends to spend the less time for a project p, the smaller the number of matching skills is between k and p. Then, in consequence, the selection probability of a pair (k, p) is the greater, the greater our expectation is that $x_{kp} = 0$ holds in a high-quality solution for the underlying problem.

Note that surrogate objective function (6.22) and similar surrogate objective functions must be applied with great care if the remaining linear programs $\mathrm{LP}_t^{\mathrm{rem}}$, $t \in \mathcal{T}$, are formulated as generalized minimum cost flow problems; we recommend to abstain from the use of surrogate objective functions in this case. The reason for this limitation is that the generalized minimum cost flow formulation of a remaining linear program does not *explicitly* demand that project requirements must be satisfied. If no surrogate objective function is applied and if the satisfaction of all project requirements is possible, the satisfaction is achieved, because the specific configuration of arc costs implicitly demands the fulfillment of project requirements. The specific configuration of arc costs guarantees that total flow costs are the less, the more flow reaches the terminal node of the flow network. A surrogate objective function such as (6.22) modifies the costs of the arcs that run somewhere between the source node and the terminal node of the flow network. This modification can imply that a flow which is not a maximum flow is cheaper than any maximum flow. Hence, if we apply (6.22) in conjunction with a generalized minimum cost flow network model, we may get wrong answers to the question whether an assignment $\boldsymbol{x}$ is feasible or not. The following example demonstrates how modified arc costs can lead to a wrong answer.

Example 6.12 Consider the following instance with $\mathcal{K} = \{k_1, k_2\}$, $\mathcal{D} = \{d\}$, $\mathcal{P} = \{p_1, p_2\}$, $\mathcal{S} = \{s_1, s_2, s_3\}$, and $\mathcal{T} = \{t\}$. Project p_1 requires only skill s_1, whereas project p_2 requires only the skills s_2 and s_3. The skill requirements are given by $r_{pst} = 2$, $p \in \mathcal{P}$, $s \in \mathcal{S}_p$. Worker k_1 masters all three skills, his respective skill levels are $l_{k_1 s_1} = 2$ and $l_{k_1 s_2} = l_{k_1 s_3} = 0.5$. Worker k_2 masters the skills s_1 and s_2 with $l_{k_2 s_1} = l_{k_2 s_2} = 1$. The availabilities are $R_{k_1 t} = 8$ and $R_{k_2 t} = 1$. Assume that $rd_{dt} = 0$ holds, that each worker is assigned to each project, and that the corresponding remaining linear program of period t is formulated as a generalized minimum cost flow problem.

Let the contributions of workers be associated with costs according to (6.22) and let the costs for one flow unit on the backward arc from the terminal node to the source node be -1.1 in order to ensure that each flow that circulates through the network has negative cost. There exist feasible solutions of the remaining LP, i.e., feasible flows that saturate the arcs that are associated with project requirements. A minimum cost solution among the feasible solutions incurs costs of -3.225 and is given by $y_{k_1 p_1 s_1 t} = 1$, $y_{k_1 p_2 s_2 t} = 3$, $y_{k_1 p_2 s_3 t} = 4$, and $y_{k_2 p_2 s_2 t} = 0.5$, for example. However, there are feasible network flows that do not correspond to feasible solutions of the remaining LP but that incur less costs than any feasible solution. The minimum cost flow accrues cost of -3.5 and is specified by $y_{k_1 p_2 s_2 t} = y_{k_1 p_2 s_3 t} = 4$ and $y_{k_2 p_1 s_1 t} = 1$. Note that this flow does not completely satisfy the requirement of project p_1 for skill s_1.

For the instance in this example, it would be necessary to set the costs per unit flow on the backward arc to a value less than -1.375 in order to ensure that every minimum cost flow is a maximum flow. □

The second property of DROP concerns the ability of this solution method to find an existing feasible solution and to find an existing optimal solution. DROP has this property regardless of whether a surrogate objective function is applied or not. If a feasible solution exists for an instance, DROP finds a feasible solution, because the start solution, where each worker is assigned to every suitable project, must be feasible. If no feasible solution

exists for an instance, DROP approves that no feasible solution exists, as soon as DROP has confirmed that the start solution $\boldsymbol{x} = \mathbf{1}$ is infeasible.

Furthermore, DROP can find an optimal solution for any instance if a feasible solution exists. Though, this statement is only valid if DROP is executed without the extension outlined in Algorithm 6.13, i.e., if we do not cancel assignments without contribution outside of drop operations, and if the selection probabilities of all pairs $(k, p) \in \mathcal{C}$ are always positive. If these conditions hold, all feasible solutions can be constructed by DROP, because all sequences of pairs (k, p) in $\mathcal{C}$ can be generated, since a sequence is generated randomly.

The ability of DROP to find an existing feasible solution distinguishes DROP from GRAP and ISAP. The latter two heuristics can fail to find an existing feasible solution, whereas DROP always finds an existing feasible solution. As an example for an instance where GRAP and ISAP fail, but DROP succeeds, consider the instance that we described in the proof of Proposition 6.1 on page 134.

6.2.4 A rounding procedure (ROUND) and a relax-and-fix approach

In this subsection, we present a *rounding procedure* for the workforce assignment problem. We call this procedure ROUND. Additionally, we will sketch and discuss a relax-and-fix approach.

The basic idea of ROUND is to solve the LP relaxation of the workforce assignment problem and to round for each project p a large fractional variable x_{kp} to 1. All variables that are rounded up to 1 are fixed to this value before the LP relaxation is solved again. Solving the LP relaxation and rounding and fixing variables is repeated until a solution is obtained in which all variables x_{kp} are integer.

In the following, we will first classify rounding heuristics as MIP-based heuristics. Next, we briefly report on applications of rounding heuristics that are documented in the literature, before we outline our heuristic ROUND, succinctly describe a multi-start version of ROUND, and highlight some properties of ROUND. Finally, we sketch and discuss a relax-and-fix approach to our problem.

Rounding heuristics, which are also called dive-and-fix heuristics, belong to the class of MIP-based heuristics (cf. Wolsey, 1998, pp. 214–217). MIP-based heuristics are heuristics that solve relaxations of the original MIP several times. Each time after having solved a relaxation, the solution is exploited to obtain a relaxation with a reduced search space. A MIP-based heuristic terminates when the solution to a relaxation is feasible for the original MIP or when the reduced search space is empty and no feasible solution can be returned. An empty search space can occur in the course of a MIP-based heuristic, although a feasible solution for the original MIP may exist. If the relaxation that is repeatedly solved is the LP relaxation, the heuristic is also called LP-based heuristic. Our rounding procedure is an LP-based heuristic.

The rationale for the use of MIP-based heuristics is the fact that they exploit the power of mathematical programming by solving relaxations. An optimal solution for the relaxation of a problem can often provide insight in the shape of an optimal solution of the original problem. If the LP relaxation is considered, the shape of an optimal integer-feasible solution is the clearer the tighter the relaxation is, i.e., the closer the feasible region of the LP relaxation comes to the convex hull of the original problem. When the

feasible region of the LP relaxation coincides with the convex hull, an optimal solution for the LP relaxation is also an optimal solution for the original MIP. As we tightened our two models of the workforce assignment problem by valid inequalities, we were optimistic that an LP-based rounding procedure may determine solutions of good quality.

Rounding heuristics have been successfully applied in order to solve MIPs in different fields such as capacitated lot-sizing (cf. e.g. Maes et al., 1991; Alfieri et al., 2002), facility location (cf. Bloemhof-Ruwaard et al., 1996), and data association, i.e., matching elements of two data sets (cf. Zhang et al., 2005). Wallace (2010) outlines a very simple, but fast rounding heuristic that was integrated in a MIP solver to determine upper (lower) bounds for minimization (maximization) problems. Though, his heuristic is not appropriate for our problem, because the simplicity of his heuristic would lead to solutions of poor quality for our problem.

In the field of workforce management, the following two publications amongst others feature rounding heuristics. Fowler et al. (2008) designed two rounding heuristics to solve a multi-period problem where for each period the number of workers that are hired, fired, and cross-trained must be determined such that total costs are minimized. In each period and for each skill s, a demand for workers that master skill s is given. If the demand is not satisfied, penalty costs are incurred. The two rounding heuristics are compared to a genetic algorithm. The rounding heuristics clearly outperform the genetic algorithm with respect to solution time, while solution quality is almost the same.

Miller and Franz (1996) consider a multi-period assignment problem where one task must be assigned to each worker in each period and several constraints must be observed. In their approach, Miller and Franz (1996) carefully select binary variables with fractional values in the solution of the LP relaxation and round these variables up to 1. A conservative selection rule is applied in order to avoid infeasibility. For our problem, however, rounding up a fractional variable x_{kp} is always feasible.

In the references for rounding heuristics that we just mentioned, deterministic procedures are applied. Though, also randomized rounding procedures have been developed (cf. Bertsimas and Vohra, 1998). An example for a deterministic procedure for a problem with binary variables is a procedure with the rule to round up a fractional variable x if $x \geq 0.5$, and to round down x otherwise. In case of a randomized rounding procedure, a probability of rounding up the variable x is determined. This probability usually depends on the value of x. For example, the probability of rounding up (down) increases with increasing (decreasing) value of x. Our procedure ROUND is a mixture of deterministic and randomized rounding. Incorporating randomness in ROUND facilitates a multi-start procedure.

Both rounding and relax-and-fix heuristics are applied by Kimms and Müller-Bungart (2007) to a problem where a television channel must select orders for advertising spots and where the spots of selected orders must be assigned to commercial breaks.

All the references mentioned show that rounding heuristics can be applied to different problems that can be modeled as a MIP. Though, adjustments of the simple basic scheme of a rounding heuristic are necessary in order to cope with special properties and constraints of the problems. Only such adjustments lead to a promising solution method.

Many possibilities exist to design a rounding heuristic for the workforce assignment problem. The manifold possibilities result from the manifold answers that can be given to design questions such as “How many variables shall be rounded in an iteration?” and

"Which variables shall be rounded (with which probability)?" We implemented and tested several variants. The variant which we present here and which we call ROUND performed best with respect to average solution quality for our test instances.

Algorithm 6.26 summarizes the heuristic ROUND. The heuristic starts by initializing for each project p the set $\mathcal{K}_p^{\text{frac}}$. This set contains every worker k who is suitable for project p and whose corresponding variable x_{kp} has not been fixed to either 0 or 1 so far, i.e., whose variable x_{kp} can be fractional in the solution of the LP relaxation. At the outset, no variable x_{kp}, $k \in \mathcal{K}_p^{\text{suit}}$, is fixed and the set $\mathcal{K}_p^{\text{frac}}$ contains *all* workers who are suitable for project p.

After this initialization, the LP relaxation of the original MIP is solved. The LP relaxation in ROUND is based on a tightened version of the standard or the network model. To clarify the term *tightened version*, we describe the tightened version of the network model. To tighten the network model (4.19)–(4.29), big-M constraints (4.24) were replaced by big-M constraints (6.5) and big-M constraints (6.6) were added. While big-M constraints (6.5) restrict all variables f_{kpt}^{proj}, big-M constraints (6.6) restrict all variables y_{kpst}. Furthermore, valid inequalities (6.1) and (6.3) were added. Valid inequalities (6.1) require for each project $p \in \mathcal{P}$ and each skill $s \in \mathcal{S}_p$ that a minimum number of workers who master skill s must be assigned to project p. Valid inequalities (6.3) require for each project $p \in \mathcal{P}$ that a minimum number of workers who can contribute to project p must be assigned to project p. The tightened version of the standard model is created analogously.

The solution $(\boldsymbol{x}, \boldsymbol{y})$ of the LP relaxation is stored and analyzed. If all variables x_{kp} are integer in this solution, an optimal solution for the original MIP has been found and Algorithm 6.26 terminates. If there is at least one fractional variable x_{kp} in the solution of the LP relaxation, the iterations of rounding variables, fixing them, and solving the new LP relaxation are started. These iterations are continued until a solution of the LP relaxation is reached in which all variables x_{kp} are integer. This integer-feasible solution can be an optimal solution for the original MIP, but it need not.

At the beginning of each iteration, after solving the current LP relaxation, each variable x_{kp} that has not been fixed and that takes on an integer value in the current solution is fixed to this value (cf. lines 11–15 of Algorithm 6.26). Fixing these variables reduces the size of the LP relaxation and, thus, speeds up the solution procedure. Each variable x_{kp} that was fixed to its current integer value is removed from the corresponding set $\mathcal{K}_p^{\text{frac}}$.

We round up at most one fractional variable per project in each iteration. All projects p for which variables x_{kp} with non-integer values exist in the current solution are gathered in the set $\mathcal{P}^{\text{toRound}}$. For each project p in this set, we will round up a fractional variable and fix it to 1 if some conditions are met. We will specify these conditions later. If the conditions are not met, no variable that is associated with project p will be rounded up and fixed. Hence, for each project $p \in \mathcal{P}^{\text{toRound}}$ at most one fractional variable that is associated with project p is rounded up and fixed. No more than one variable is rounded up and fixed, because rounding up and fixing one variable may drive all other fractional variables that are associated with project p to 0 in the solution of the new LP relaxation.

For each project p in $\mathcal{P}^{\text{toRound}}$, the largest and the second largest fractional variable associated with p are considered for rounding up, but at most one of them is actually rounded up. The projects in $\mathcal{P}^{\text{toRound}}$ are considered in random order. The probability for being the project considered next is set to $\frac{1}{|\mathcal{P}^{\text{toRound}}|}$ for each project. Assume that

Algorithm 6.26 The heuristic ROUND

Input: Instance data
Output: A feasible solution $(\boldsymbol{x}, \boldsymbol{y})$ for the workforce assignment problem

for all $p \in \mathcal{P}$ **do**
    $\mathcal{K}_p^{\text{frac}} := \mathcal{K}_p^{\text{suit}}$;
Solve the LP relaxation;
Store the values of all variables x_{kp} and y_{kpst};
while $\exists (k,p) \mid x_{kp} \notin \{0,1\}$, $p \in \mathcal{P}$, $k \in \mathcal{K}_p^{\text{frac}}$ **do**
    $\mathcal{P}^{\text{toRound}} := \emptyset$;
    $\mathcal{K}^{\text{rounded}} := \emptyset$;
    for all $p \in P \mid \mathcal{K}_p^{\text{frac}} \neq \emptyset$ **do**
        for all $k \in \mathcal{K}_p^{\text{frac}}$ **do**
            if x_{kp} is equal to 0 or 1 **then**
                Fix x_{kp} to its current value;
                $\mathcal{K}_p^{\text{frac}} := \mathcal{K}_p^{\text{frac}} \setminus \{k\}$;
        if $\mathcal{K}_p^{\text{frac}} \neq \emptyset$ **then**
            $\mathcal{P}^{\text{toRound}} := \mathcal{P}^{\text{toRound}} \cup \{p\}$;
    while $\mathcal{P}^{\text{toRound}} \neq \emptyset$ **do**
        Randomly select a project p from the set $\mathcal{P}^{\text{toRound}}$;
        $\mathcal{P}^{\text{toRound}} := \mathcal{P}^{\text{toRound}} \setminus \{p\}$;
        $k^{\text{toRound}} := -1$;
        $k' := \arg\max_{k \in \mathcal{K}_p^{\text{frac}}} x_{kp}$;
        if $k' \notin \mathcal{K}^{\text{rounded}}$ **then**
            $k^{\text{toRound}} := k'$;
        if $\left|\mathcal{K}_p^{\text{frac}}\right| > 1$ **then**
            $k'' := \arg\max_{k \in \mathcal{K}_p^{\text{frac}} \setminus \{k'\}} x_{kp}$;
        if $k'' \notin \mathcal{K}^{\text{rounded}}$ **then**
            if $k^{\text{toRound}} = -1$ **then**
                $k^{\text{toRound}} := k''$;
            else
                if $\text{rand}(0,1) \leq \frac{x_{k''p}}{x_{k'p} + x_{k''p}}$ **then**
                $k^{\text{toRound}} := k''$;
        if $k^{\text{toRound}} \neq -1$ **then**
            Fix $x_{k^{\text{toRound}}p}$ to 1;
            $\mathcal{K}_p^{\text{frac}} := \mathcal{K}_p^{\text{frac}} \setminus \{k^{\text{toRound}}\}$;
            $\mathcal{K}^{\text{rounded}} := \mathcal{K}^{\text{rounded}} \cup \{k^{\text{toRound}}\}$;
    Solve the LP relaxation;
    Store the values of all variables x_{kp} and y_{kpst};

project p is selected from the set $\mathcal{P}^{\text{toRound}}$ for being considered next. For rounding up and fixing to 1, we consider the largest fractional variable $x_{k'p}$ that is associated with project p,

and the second largest fractional variable $x_{k''p}$ if there is more than one fractional variable associated with project p. Ties are broken by the smallest-subscript rule with respect to subscript k.

However, variable $x_{k'p}$ comes into question for rounding up and fixing only if no other variable $x_{k'p'}$, $p' \in \mathcal{P}_{k'}^{\text{suit}} \cap \mathcal{P}^{\text{toRound}}$, that is associated with worker k' has already been rounded up and fixed in this iteration. The same holds for the variable $x_{k''p}$. To check for a worker k if a variable $x_{kp'}$, $p' \in \mathcal{P}_{k}^{\text{suit}} \cap \mathcal{P}^{\text{toRound}}$, has already been rounded up and fixed for this worker in the current iteration, the set $\mathcal{K}^{\text{rounded}}$ is scrutinized. This set records the indices of all workers for whom a variable has been rounded up and fixed to 1 in the current iteration.

In the following description of the actual rounding operation, we consider the most general case: Let there be two fractional variables that are associated with project p in the current solution, namely, $x_{k'p}$ and $x_{k''p}$ with $x_{k'p} \geq x_{k''p}$. If both workers k' and k'' are in the set $\mathcal{K}^{\text{rounded}}$, no variable is rounded up and fixed for project p. If only one of these workers belongs to the set $\mathcal{K}^{\text{rounded}}$, we round up and fix the variable that is associated with the other worker that is not in $\mathcal{K}^{\text{rounded}}$. The worker for which the corresponding variable is rounded up and fixed to 1 is denoted by k^{toRound}.

If both variables $x_{k'p}$ and $x_{k''p}$ come into question for rounding up and fixing to 1, we randomly select one of them. The probability of selecting variable $x_{k'p}$ ($x_{k''p}$) takes into account the value of $x_{k'p}$ ($x_{k''p}$) in comparison to the value of $x_{k''p}$ ($x_{k'p}$); selection probabilities are given by $\frac{x_{k'p}}{x_{k'p}+x_{k''p}}$ and $\frac{x_{k''p}}{x_{k'p}+x_{k''p}}$, respectively. A random number that is uniformly distributed in the interval $[0, 1)$ and generated by the function rand$(0, 1)$ decides according to the probabilities whether $x_{k'p}$ or $x_{k''p}$ is rounded up and fixed. The worker whose variable is selected for rounding up is also denoted by k^{toRound}.

If a variable $x_{k^{\text{toRound}}p}$ was specified for project p, this variable is rounded up and fixed to 1 (cf. lines 33–34). Worker k^{toRound} is removed from the set $\mathcal{K}_p^{\text{frac}}$ and added to the set $\mathcal{K}^{\text{rounded}}$ in order to prevent that another fractional variable that is associated with worker k^{toRound} is rounded up and fixed in this iteration.

When all projects in $\mathcal{P}^{\text{toRound}}$ have been considered, the new LP relaxation is solved and the new solution values are stored. The old values are overwritten. If the new solution is not integer-feasible, the next iteration is started.

The variables y_{kpst} that correspond to variables x_{kp} that were fixed to 1 are not fixed until the very end of ROUND. As in DROP, this flexibility of the variables $\boldsymbol{y}$ allows the LP solver to adjust the values of these variables to changes in $\boldsymbol{x}$.

For a multi-start version, we embedded ROUND in the generic multi-start procedure outlined in Algorithm 6.5 on page 126. This multi-start version of ROUND constructs different solutions, because projects are considered in random order in each iteration within a pass of ROUND and variables for rounding up and fixing are selected randomly from a candidate set that contains up to two candidates. However, like in DROP, we do not build a solution from scratch in each pass of the multi-start method. To save time, we solve the initial LP relaxation only once and fix all variables x_{kp} that take on an integer value in the solution of this relaxation to the respective value. The resulting solution, which is integer-infeasible in general, is stored and used as the starting point for each pass of the multi-start method. Hence, in each pass, some variables x_{kp} are kept fixed to the same value as in all previous passes.

That some variables are fixed during the multi-start procedure implies a small degree

of rebuild according to Martí et al. (2010, p. 274). The actual purpose of a certain degree of rebuild is not to save time but to start each pass from a promising point. In our case, beginning each pass with some variables already fixed did save time, but also comes along with the danger of a drop in solution quality, because fixing a variable to the same value in each pass constrains the solution space that is explored. However, preliminary tests have shown that solution quality did not decline in our case.

In the following, four properties of the heuristic ROUND shall be discussed. First, the heuristic ROUND excludes that in an iteration more than one fractional variable that is associated with a worker k is rounded up and fixed. The advantage of this cautious approach can be demonstrated by the following example. Let $x_{kp'}$ and $x_{kp''}$ be two variables that are associated with worker k and that take on fractional values in the solution of the LP relaxation. Rounding up at most one fractional variable per worker k, say variable $x_{kp'}$, enables a reallocation of workload such that the time of worker k may be completely occupied by working for project p', while $x_{kp''}$ becomes 0. Hence, it may be detrimental to round up and fix both $x_{kp'}$ and $x_{kp''}$ to 1. This is why at most one variable is rounded up per worker in each iteration of ROUND.

Second, ROUND does not only consider the largest fractional variable that is associated with project p for rounding up, but also the second largest fractional variable. This may be detrimental if only one solution is constructed. Though, if ROUND is embedded in a multi-start procedure, this variant of randomized rounding examines a greater part of the solution space and leads to better results, as preliminary tests revealed.

Third, ROUND finds always a feasible solution for the original MIP. Since a positive variable x_{kp} is never rounded down to 0, the way to a feasible solution is never obstructed. Only variables that are equal to 0 in a solution for the LP relaxation are fixed to 0. Fixing a variable to 0 that is equal to 0 in the solution of the LP relaxation does not jeopardize feasibility, because rounding up all variables x_{kp} for which $x_{kp} > 0$ holds in the current solution provides always an integer-feasible solution.

Fourth, ROUND can fail to find an optimal solution even if ROUND is executed an infinite number of times within a multi-start procedure. Three scenarios may demonstrate how a way to an optimal solution is obstructed in the course of ROUND. Firstly, fixing a variable x_{kp} to 0 for which $x_{kp} = 0$ holds in the solution of the initial LP relaxation means that an optimal solution in which $x_{kp} = 1$ holds cannot be reached anymore. This optimal solution might have been reached if x_{kp} had not been fixed to 0 but had been allowed to take on a positive value in later iterations. Secondly, likewise, fixing a variable x_{kp} to 1 that is equal to 1 in the initial solution of the LP relaxation obstructs the way to an optimal solution if $x_{kp} = 0$ holds in every optimal solution. If other variables than x_{kp} were fixed to 1, the value of x_{kp} might have dropped down to 0 in a later iteration. Thirdly, if there are more than two fractional variables x_{kp} that are associated with a project p in the solution of the initial LP relaxation and the variables related to the other projects are integer, the consideration of the two largest variables associated with p for rounding up and fixing to 1 may render it impossible to reach an optimal solution if both considered variables equal 0 in every optimal solution.

To overcome the drawback that ROUND can fail to find an optimal solution even when a multi-start procedure is used, the subsequent modifications would be necessary. A variable must never be fixed to 0 and every variable that has not been fixed so far must be considered as a candidate for rounding up and fixing to 1. Consequently, variables whose

value is equal to 0 in the solution of an LP relaxation are candidates and variables whose value is fractional or equal to 1 in the solution of an LP relaxation are also candidates. From all these candidates, only one variable is randomly selected for rounding up and fixing to 1 in each iteration. These modifications would enable ROUND to find an optimal solution. Though, the information contained in the solution of the LP relaxation, i.e., the information about the shape of an optimal solution for the original MIP, would be ignored. The application of the modifications has two consequences. First, computation times increase, because each time after solving the LP relaxation only one variable is rounded up and fixed. Second, the expected solution quality given a limited number of passes would deteriorate compared to ROUND. This is why these modifications have virtually no practical benefit.

Eventually, let us turn to relax-and-fix heuristics and let us consider our relax-and-fix heuristic for the workforce assignment problem. Like rounding heuristics, relax-and-fix heuristics belong to the class of MIP-based heuristics and exploit the power of mathematical programming.

The basic idea of a relax-and-fix heuristic for a PIP or MIP is to relax integrality constraints only for some integer variables in each iteration, while other integer variables are considered as truly integer such that a reduced MIP can be solved in each iteration. To be more precise, integer variables are divided into three groups in each iteration. The first group comprises variables for which the integrality constraints are relaxed, whereas the integrality constraints must be observed for the variables of the second group. The third group contains those variables that are finally fixed. This group is empty at the beginning of a relax-and-fix procedure, when no variable is fixed. These three groups are associated with a reduced MIP that has less truly integer variables than the original MIP. When the reduced MIP has been solved, the variables of the second group are fixed to their solution values and are transferred to the third group. For the next iteration, some variables of the first group are transferred to the second group. Two outcomes are possible for a relax-and-fix heuristic. Either an integer-feasible solution is obtained at the latest when all integer variables have been transferred to the third group or at some point no feasible solution can be found for the reduced MIP, although a feasible solution may exist for the original problem.

Relax-and-fix heuristics and variants thereof have been successfully applied in order to solve MIPs in different fields, especially in the field of capacitated lot-sizing (cf. Dillenberger et al., 1994; Sürie and Stadtler, 2003; Förster et al., 2006; Sahling, 2010). In the area of project management, Escudero and Salmeron (2005) apply a relax-and-fix approach to a problem that comprises project selection and scheduling of project start times. Binary variables x_{pt} are used for modeling. A variable x_{pt} equals 1 if project p is selected and started in period t. Escudero and Salmeron (2005) examine different strategies that differ with respect to the question in which iteration which of the binary variables belong to the second group. The binary variables of the second group are considered as truly binary variables. One strategy assigns for each iteration the binary variables that are associated with a certain period to the second group. The periods are considered in chronological order. In another strategy, the second group contains those variables whose objective function coefficients lie in a certain interval. The intervals are considered in the order of decreasing attractiveness with respect to the objective function. Finally, as al-

ready mentioned, relax-and-fix heuristics (and rounding heuristics) are applied by Kimms and Müller-Bungart (2007) to a problem of selecting and scheduling advertising spots.

At the beginning of our relax-and-fix procedure, the LP relaxation of the original MIP is solved and all integer-valued variables x_{kp} are fixed to their current value. Then, in each upcoming iteration of this procedure, a reduced MIP is solved. In each iteration, the binary variables of only one project p are considered as truly binary variables (variables in the second group), while all other binary variables are relaxed or have already been fixed (variables in the first or third group, respectively). A solver is invoked to solve the reduced MIP within a prescribed time limit. If no integer-feasible solution for the reduced MIP is found, the relax-and-fix heuristic terminates without returning a solution.

If a solution for the reduced MIP is found, the variables x_{kp} associated with project p are fixed to their values in the current solution. For the next iteration, the variables $x_{kp'}$ of another project p' are transferred from the first to the second group and thus are considered as truly binary. The order of projects in which the corresponding variables are considered as truly binary is randomly determined. When all variables x_{kp} have been fixed, the relax-and-fix procedure terminates. The variables y_{kpst} are never fixed during the procedure but at the very end when an integer-feasible solution has been found.

The outcomes of preliminary tests of our relax-and-fix procedure were not promising. If the time granted for solving the reduced MIPs was short, solution quality was poor. Sometimes, feasible solutions for a reduced MIP were not found within the time limit. In this case, the procedure could not provide a solution and was aborted. When more time was granted, solution quality improved. Though, the increase in time that was required to obtain competitive solutions for large-sized instances was prohibitive. This is why we abandoned the relax-and-fix approach and do not consider this approach in the performance analysis in the next chapter.

Chapter 7

Numerical analysis

In this chapter, we will report how we generated test instances, how we used them to test the solution methods, and what results we obtained. The tests had three major aims. First, we wanted to find out up to which instance sizes exact methods could provide acceptable solutions in acceptable time for the project selection problem and for the workforce assignment problem, which are strongly NP-hard. Second, we wanted to reveal the impact of certain parameters on solution time and quality. Among these parameters are the number of workers and the ratio of project workload to the availability of the workforce. Third, we wanted to assess the four heuristics that we devised for the workforce assignment problem and we wanted to conclude which one is the most suitable.

We implemented all solution methods in the C++ programming language, which is a widely used object-oriented programming language (cf. Josuttis, 2001, 2002). The program code was compiled using the C++ compiler of Microsoft Visual Studio 2010 Premium. For solving optimization problems with the LP solver or the MIP solver of IBM ILOG CPLEX, we used the CPLEX Concert Technology C++ library. This library offers classes and functions to implement an optimization model in C++ and provides an interface to the CPLEX solvers. We used the Concert Technology library and the solvers of CPLEX 12.4 (cf. IBM, 2011).[1] Data input and output is handled by the C++ programs via text files. A parser reads in instance data from a text file and results are written to another text file.[2]

We wrote a C++ program to automatically generate test instances for the project selection problem and the workforce assignment problem. This program allows to generate instances of various sizes and with various characteristics for systematic tests. The program, called *instance generator*, will be described in Section 7.1 in more detail. For the utilization leveling problem, we used instances of the workforce assignment problem together with corresponding solutions as input data.

All programs were executed on personal computers that operated under Microsoft

[1] The optimization models for our three problems as well as Algorithm 6.4 for the utilization leveling problem were also implemented in the modeling system GAMS (cf. GAMS Development Corporation, 2013). The code required to implement optimization models in GAMS is quite simple and easier to maintain than C++ code. Furthermore, GAMS allows to switch from one solver to another without great effort.

[2] We also developed an application with Microsoft Access based on Visual Basic for Applications (cf. Albrecht and Nicol, 2007). This application provides a graphical user interface and calls GAMS to solve our three optimization problems. GAMS, in turn, reads instance data via SQL queries from an Microsoft Access database, invokes CPLEX for solving MIPs, and writes results to the database. The Microsoft Access based application can easily be integrated within the software environment of many firms. The application was developed to serve as a vehicle for testing and discussing with practitioners.

M. Walter, *Multi-Project Management with a Multi-Skilled Workforce,* Produktion und Logistik, DOI 10.1007/978-3-658-08036-5_7, © Springer Fachmedien Wiesbaden 2015

Windows 7 and that featured an Intel Core i7 processor with 4 cores. Each core had a clock rate of 3.6 GHz and could handle 2 threads. Every computer was equipped with 32 GB RAM. The algorithms that we devised use only 1 thread, as they are not parallelized. The solvers of CPLEX 12.4, however, can use all 8 threads. As we used a 32 bit version of CPLEX, memory was limited to 4 GB RAM. In order to illustrate the potential of CPLEX, we will also present selected results for a 64 bit version of CPLEX, which could exploit 32 GB RAM.

For each of our three problems, we will describe the sets of test instances and the results that the different solution methods provided for these test sets. In Section 7.2, test sets and results for the project selection problem are presented, before the instances and the corresponding results for the workforce assignment problem are outlined in Section 7.3. Finally, test sets for the utilization leveling problem and the performance of our solution methods for this problem will be presented in Section 7.4.

7.1 Generation of test instances

In this section, we will describe how we created instances for our three problems in order to test our solution methods. At the beginning, we will briefly discuss two alternative methods to obtain instance data, namely, (1) gathering data from real-life cases and (2) randomly generating coherent artificial data. We chose the second method and wrote a program that generates artificial instances for the project selection and the workforce assignment problem. Henceforth, this program will be called *instance generator*. We will describe the input parameters of the instance generator, which define key characteristics of the instances to generate, and we describe how instance data such as project requirements and skill levels of workers are determined by the instance generator. We will specify the values (default values) of those input parameters that were kept fixed for all test sets (most of the test sets) presented in this chapter. For a resulting instance, we will define a measure that relates the size of a skill requirement to the availability of those workers that master the corresponding skill.

In general, two distinct sources for instance data exist. Instance data can be obtained from real-life problems or can be fictitious (cf. Salewski et al., 1997, pp. 102–103). For the case of fictitious data, we assume that an instance is systematically constructed with the help of random number generators.

Both real-life instances and artificial instances have advantages and disadvantages. While real-life data represent real problems for which our solution methods are intended, artificial data may not provide a realistic picture of practical cases. However, it is costly and of limited use to collect data from real-life cases. It is costly, because in a firm data may not have been recorded or may be scattered across several departments. Even if data of a firm are easily accessible, they are of limited use, because the corresponding records usually provide data of one instance per year. Since data records may go back only some years, there is a limited number of instances per firm and these instances tend to closely resemble one another. Even if instances of several firms can be obtained, it is highly improbable that the instances are representative for the complete set of real cases. Hence, results of a numerical analysis that is based on a limited set of real-life instances can hardly be generalized.

A systematic generation of artificial instances, on the other hand, is inexpensive, fast, and can provide a large set of diverse instances that facilitate a thorough testing of solution methods. An appropriate set of artificial instances can reflect the range of instances that exist and arise in practice across hundreds of firms. Thus, in our opinion, reasonably constructed artificial instances are more suitable for testing solution methods. Consequently, we decided to generate artificial instances.

Nevertheless, it is fruitful and necessary to get insight into real-life instances in order to construct fictitious instances reasonably. For the IT center that triggered our research, we got information about characteristics of its project selection problem and its workforce assignment problem. We used this information to design our instance generator.

The instance generator has 22 main input parameters, which are specified in Table 7.1. In this table, also default values are stated for those parameters that were kept fixed for generating most of our test instances. We will explain the parameters and their role during instance construction in the following paragraphs.

We begin with the first five input parameters in Table 7.1 to give a first impression of the operating principle of the instance generator. Assume that we choose $K = 40$, $D = 5$, $\hat{P} = 20$, $S = 8$, and $T = 12$ for the first five input parameters. Then, the instance generator outputs a desired number of instances that feature 40 workers partitioned into 5 departments, 20 projects, 8 skills that are mastered by the workforce in total, and 12 periods. While the numbers of workers, departments, projects, skills, and periods are the same for all instances of the generated test set, data such as skills mastered by a single worker, skill levels, and project requirements are randomly generated and vary from instance to instance. In the following, we will explain the other main input parameters and elaborate on how the random data are generated. Whenever we state in the description of the instance generator that a value is randomly determined, the value is drawn from a discrete uniform distribution, unless otherwise indicated. For all generated test instances, the planning horizon was set to $T = 12$.

For each department $d \in \mathcal{D}$, a random number of members $|\mathcal{K}_d|$ is determined such that the equation $\sum_{d\in\mathcal{D}} |\mathcal{K}_d| = K$ holds. For department $d = 1, \ldots, D-1$, the instance generator randomly chooses the value of $|\mathcal{K}_d|$ from the discrete set $\big\{\text{round}(0.8 \cdot \frac{K}{D}), \text{round}(0.8 \cdot \frac{K}{D}) + 1, \text{round}(0.8 \cdot \frac{K}{D}) + 2, \ldots, \text{round}(1.2 \cdot \frac{K}{D})\big\}$, where the function round(a) returns the integer that is closest to a. The remaining workers are assigned to department $d = D$. If at some point no workers remain that can be assigned to the next department, the procedure is repeated until a feasible partition of the workforce is reached. Finally, workers $k = 1, \ldots, |\mathcal{K}_{d_1}|$ are assigned to department $d = 1$, workers $k = |\mathcal{K}_{d_1}| + 1, \ldots, |\mathcal{K}_{d_1}| + |\mathcal{K}_{d_2}|$ are assigned to department $d = 2$, and so on.

The availabilities R_{kt}, $k \in \mathcal{K}$, $t \in \mathcal{T}$, are drawn from a discrete uniform distribution between *KminR* and *KmaxR*. We set *KminR* := 120 and *KmaxR* := 160. The latter value represents an availability of 160 hours per period, which is a typical number of contracted hours per month and corresponds to a daily working time of 8 hours. We assumed that the times for vacation, routine jobs and other individual tasks sum up to at most 40 hours per worker and month, leading to a minimum availability of 120 hours.

The assignment of skills to workers is controlled by two basic decisions. The first decision is about whether members of a department share a common skill or not. If all members master a common skill (input parameter *boolSameSkill* = true), we assign skill s_1 to each worker of department d_1 and skill s_2 to each worker of department d_2, and so

Table 7.1: Main input parameters of the instance generator and their default values

Parameter	Domain	Value*	Description
K	$\mathbb{N}$	–	Number of workers
D	$\mathbb{N}$	–	Number of departments
$\hat{P}$	$\mathbb{N}$	–	Number of projects including ongoing projects and must projects
S	$\mathbb{N}$	–	Number of skills
T	$\mathbb{N}$	12	Number of periods
KminR	$\mathbb{N}$	120	Minimum availability of a worker in a period
KmaxR	$\mathbb{N}$	160	Maximum availability of a worker in a period
boolSameSkill	{true, false}	true	Indicates whether all workers of a department master a common skill or not
boolNumSkillsRand	{true, false}	true	Indicates whether each worker masters a random number of skills or whether the number of skills mastered by a worker depends on the group he belongs to
KminS	$\mathbb{N}$	1	Minimum number of skills mastered by a worker
KmaxS	$\mathbb{N}$	3	Maximum number of skills mastered by a worker
probS1	$[0, 1]$	(0.6)	Probability for a worker to master exactly one skill (group 1)
probS2	$[0, 1]$	(0.3)	Probability for a worker to master exactly two skills (group 2)
probS3or4	$[0, 1]$	(0.1)	Probability for a worker to master either three or four skills (group 3)
ρ^{dep}	$[0, 1]$	0.2	Desired ratio of a departmental requirement to the corresponding total availability of department members
PminDur	$\mathbb{N}$	3	Minimum duration of a project
PmaxDur	$\mathbb{N}$	9	Maximum duration of a project
sharePongoing	$[0, 1]$	0	Share of ongoing projects
sharePmust	$[0, 1]$	0	Share of projects that must be selected
PminS	$\mathbb{N}$	5/–	Minimum number of skills required by a project
PmaxS	$\mathbb{N}$	7/–	Maximum number of skills required by a project
ρ^{proj}	$\mathbb{R}_{\geq 0}$	2.5/0.6	Desired ratio of total project requirements to the corresponding total availability of the workforce

*Default values used for the generation of the instances tackled in this chapter. A dash (–) means that no default value exists; bracketed values did not come into effect due to *boolNumSkillsRand* = true; for an entry a/b, a refers to instances of the project selection problem and b to instances of the workforce assignment problem.

forth.[3] The case where department members share a common skill reflects the situation in

[3]If $D > S$ and skill s_S was assigned to a department $d < D$, we assign skill s_1 to each worker of department $d + 1$, skill s_2 to each worker of department $d + 2$, and so on.

a firm that exhibits a functional structure. To give an example, let a functional structure of a manufacturing firm feature a construction department and a sales department. Each member of the sales department will typically have specific knowledge of sales operations and distribution channels. In the construction department, none of the workers may have this knowledge, or at most a few workers, e.g., due to prior job experiences. The level l_{ks} with which a member of department d masters the common skill s is randomly selected from the set $\{1, 1.5, 2\}$ for each $k \in \mathcal{K}_d$. The skill level 0.5 is not awarded, because a worker is supposed to have already gained some experience in his core area of work.

If we do not require that the members of a department share a common skill (*boolSameSkill* = false), the skill set of each worker is randomly composed. This case can reflect the situation of a consultancy firm, for example, where the departments represent the subsidiaries in different cities. Within each subsidiary, the consultants may have arbitrary skill sets.

The second basic decision is about the number of skills that are assigned to each worker. The instance generator offers two possibilities. The first possibility (input parameter *boolNumSkillsRand* = true) is that for each worker k the number of skills $|\mathcal{S}_k|$ that he masters is randomly selected from the set $\{KminS, \ldots, KmaxS\}$. Then, $|\mathcal{S}_k|$ skills are randomly selected from the set $\mathcal{S}$ and assigned to worker k. If a skill shared by all department members has already been assigned to worker k, i.e., if *boolSameSkill* = true, only $|\mathcal{S}_k| - 1$ additional skills will be selected. For each (additional) skill s, the skill level l_{ks} is randomly selected from the set $\{0.5, 1, 1.5, 2\}$.

The second possibility (*boolNumSkillsRand* = false) pursues the purpose to meet the conditions that prevail in the IT center that instigated our research. That is why this second possibility was implemented only for the case where *boolSameSkill* = true holds, i.e., where workers within a department share a common skill, as it is the case in the IT center. The prerequisite *boolSameSkill* = true is not a severe limitation, though, because the distribution of skills in the IT center reflects the situation in many firms with a functional structure. In the IT center, there are three groups of workers: some workers master only one skill (group 1), other workers master two skills (group 2), while the remaining workers master either three or four skills (group 3). These cardinalities of skill sets are typical for many firms with a functional structure if skills are defined in an aggregate fashion. To reflect these three groups, the parameters *probS1*, *probS2*, and *probS3or4* determine the probability with which each worker is assigned to the respective group. Hence, the expected share of workers in the total workforce that master only one skill is equal to *probS1*. The expected shares of workers who belong to group 2 and 3 are given by *probS2* and *probS3or4*, respectively. The sum of these three parameters is supposed to equal 1. We chose *probS1* := 0.6, *probS2* := 0.3, and *probS3or4* := 0.1 whenever we opted for the second possibility.

Since the second possibility presumes that each worker masters the skill that is associated with his department, only the additional skills and the corresponding skill levels for the workers of group 2 and 3 must be determined. For each worker who belongs to group 2 or 3 and whose first skill is skill s, the second skill is either skill $s+1$ or skill $s-1$, i.e., the second skill is a skill of a neighboring department.[4] This choice of the second skill reflects the situation that the competencies of multi-skilled workers tend to lie in related areas. For example, a member of the sales department may also be deployable

[4]For skill $s = 1$, we set $s - 1 := S$, and for skill $s = S$, we set $s + 1 := 1$.

in the marketing department and vice versa, but members of these departments may not be suited for the construction department. In this example, the sales department and the marketing department are neighboring departments. The level of the second skill is randomly chosen from the set $\{0.5, 1, 1.5, 2\}$.

Each worker k in group 3 masters three or four skills; the probability for each of the two possibilities is equal. For the third and fourth skill, we do not demand that the skills are associated with a neighboring department. The corresponding skill levels are randomly selected from the set $\{0.5, 1, 1.5\}$. Since the third and the fourth skill are likely to be farther away from the core work area of a worker in group 3, we do not award the highest skill level of 2.

After skills and skill levels have been assigned to workers, it is checked if each skill $s \in \mathcal{S}$ is mastered by at least one worker. If there is a skill that is not mastered by any worker, the instance generator stops the construction of the instance and starts a new try. If all skills are covered by the workforce, the departmental requirements are determined.

The input parameter ρ^{dep} determines the magnitude of the departmental requirements. This parameter represents for each department $d \in \mathcal{D}$ and each period $t \in \mathcal{T}$ the desired ratio of the departmental requirement rd_{dt} to the total availability of the members of department d in period t. The value of rd_{dt} is drawn from a discrete uniform distribution between $\left\lfloor 0.8 \cdot \rho^{\text{dep}} \cdot \sum_{k \in \mathcal{K}_d} R_{kt} \right\rfloor$ and $\left\lfloor \rho^{\text{dep}} \cdot \sum_{k \in \mathcal{K}_d} R_{kt} \right\rfloor$, where $\lfloor a \rfloor$ denotes the largest integer that is less than or equal to a. For generating our test sets, we set $\rho^{\text{dep}} := 0.2$. A value of 0.2 implies that the utilization of the workforce caused by departmental workload lies between 16 % and 20 %.

The generation of projects and their requirements is controlled by several input parameters. For each project p out of the $\hat{P}$ projects that are generated, a random duration dur_p between $PminDur$ and $PmaxDur$ is determined. We set $PminDur := 3$ and $PmaxDur := 9$. The benefit b_p of each project p is drawn from a discrete uniform distribution between 1 and 30. The first $\left\lfloor sharePongoing \cdot \hat{P} \right\rfloor$ projects that are generated are designated as ongoing projects, which must be continued. These projects constitute the set $\mathcal{P}^{\text{ongoing}}$. Their start period is period $t = 1$, i.e., we set $t_p^{\text{start}} := 1$ for each project $p \in \mathcal{P}^{\text{ongoing}}$. The next $\left\lfloor sharePmust \cdot \hat{P} \right\rfloor$ projects are designated as mandatory projects, which must be selected. These projects form the set $\mathcal{P}^{\text{must}}$. For the mandatory projects and all remaining projects, the respective start period of such a project p is randomly selected from the set $\{1, \ldots, T - dur_p + 1\}$. The finish period t_p^{finish} of every project p including ongoing projects results from the calculation $t_p^{\text{start}} + dur_p$ - 1. So, a project p with a duration $dur_p = 1$ that starts at the beginning of period 12 ($t_p^{\text{start}} := 12$) ends at the end of period 12 ($t_p^{\text{finish}} := 12$). For the generation of our test sets, we set $sharePongoing := 0$ and $sharePmust := 0$ as this allows a clearer presentation of results and a straightforward interpretation.

The skill requirements of all $\hat{P}$ projects are randomly determined. For each project p, the number of required skills $|\mathcal{S}_p|$ is chosen between $PminS$ and $PmaxS$. Then, $|\mathcal{S}_p|$ different skills are randomly selected from the set $\mathcal{S}$. The selected skills constitute the set $\mathcal{S}_p$. For each skill $s \in \mathcal{S}_p$, a random number of periods is determined in which the skill requirement is zero. The number of periods with zero requirement for skill s lies between 0 periods and $dur_p - 1$ periods. The periods for which we set $r_{pst} := 0$ are randomly selected from the set $\{t_p^{\text{start}}, \ldots, t_p^{\text{finish}}\}$.

Now, we will outline how the skill requirements are generated for periods t with $r_{pst} >$

0. We will only sketch the generation of these values in this paragraph and the next two paragraphs, before we will give more details in the subsequent paragraphs. The magnitude of the non-zero skill requirements of a project p is affected by two factors. The first factor is the size of project p. We distinguish three project sizes: small, medium, and large. For each project, its size is randomly determined. Based on the project size provisional values for the skill requirements r_{pst} are determined.

Then, the second factor comes into play. The second factor is the input parameter ρ^{proj} that represents for each period t the desired ratio of total project requirements of all projects in period t to the total availability of the workforce in period t. If necessary, all provisional values for the project requirements r_{pst} in period t are adjusted in order to meet the desired ratio ρ^{proj} (first adjustment). This adjustment preserves relative differences in project requirements due to project size. After the adjustment, the instance generator considers for each skill s and each period t the workload of skill s and the availability of the workers who master skill s, i.e., the instance generator computes the utilization of the workers on a per-skill basis. If this utilization exceeds the parameter ρ^{proj} for skill s in period t, the requirements r_{pst} of all concerned projects p are adjusted (second adjustment). Again, relative differences in project requirements due to project size are preserved.

The first adjustment ensures that a desired utilization of the workforce is reached. Note that the desired utilization can be an overload. An overload leads to an instance of the project selection problem. For $\rho^{\text{proj}} = 2$, for example, about one project out of two can be selected if no departmental workload must be accomplished. For $\rho^{\text{proj}} = 50$, approximately one project out of 50 can be selected. An instance with an overload would be an infeasible instance of the workforce assignment problem. The second adjustment ensures that the desired utilization is not exceeded on a per-skill basis. If a second adjustment is carried out, the utilization of the workforce decreases and can fall below the desired utilization that was reached by the first adjustment.

The provisional values for the positive project requirements are determined as follows. For each positive requirement r_{pst}, a value is drawn from the discrete uniform distribution between $\lfloor a \cdot KminR \rfloor$ and $\lfloor b \cdot KminR \rfloor$. For small-, medium-, and large-sized projects the pair of parameters (a, b) is defined as $(0.2, 0.3)$, $(0.3, 0.4)$, and $(0.4,\ 0.6)$, respectively.

For the first adjustment of the provisional values, we calculate for each period $t \in \mathcal{T}$ the current utilization $util_t^{\text{proj}}$ of the workforce by project requirements as follows:

$$util_t^{\text{proj}} := \frac{\sum\limits_{p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \hat{\mathcal{P}} \mid t_p^{\text{start}} \le t \le t_p^{\text{finish}}} \ \sum\limits_{s \in \mathcal{S}_p} r_{pst}}{\sum\limits_{k \in \mathcal{K}} R_{kt}}$$

The value $util_t^{\text{proj}}$ is only an approximation for the utilization of the workforce in period t by project work. The true utilization depends on the allocation of project workload and on the skill levels of the workers. For the calculation of $util_t^{\text{proj}}$, we implicitly assume that all skill levels are equal to 1.

If the current utilization $util_t^{\text{proj}}$ does not lie in the interval $[0.98 \cdot \rho^{\text{proj}}, \rho^{\text{proj}}]$, we adjust all project requirements r_{pst} in period t by setting $r_{pst} := \left\lfloor r_{pst} \cdot \frac{\rho^{\text{proj}}}{util_t^{\text{proj}}} \right\rfloor$.

For the potential second adjustment, we calculate for each skill s and each period t the ratio $util_{st}^{\text{proj}}$ that relates the sum of all requirements for skill s in period t to the

availability of those workers who master skill s in period t. For the availability of each worker $k \in \mathcal{K}_s$, his skill level l_{ks} is taken into account:

$$util_{st}^{\text{proj}} := \frac{\sum_{p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}} \mid t_p^{\text{start}} \leq t \leq t_p^{\text{finish}} \wedge s \in \mathcal{S}_p} r_{pst}}{\sum_{k \in \mathcal{K}_s} R_{kt} l_{ks}}$$

If $util_{st}^{\text{proj}}$ is greater than ρ^{proj}, we adjust the requirements for skill s in period t by setting $r_{pst} := \left\lfloor r_{pst} \cdot \frac{\rho^{\text{proj}}}{util_{st}^{\text{proj}}} \right\rfloor$.

In the final part of the construction of an instance, some workers are assigned to each ongoing project if there are any, i.e., each set $\mathcal{K}_p^{\text{assigned}}$, $p \in \mathcal{P}^{\text{ongoing}}$, is composed. This composition works for an ongoing project p as follows. For each skill $s \in \mathcal{S}_p$, a worker k is randomly selected from the set $\mathcal{K}_s$ and added to the set $\mathcal{K}_p^{\text{assigned}}$. Since the same worker can be selected for different skills s and s' that are required by project p, $|\mathcal{S}_p|$ workers or less form the set $\mathcal{K}_p^{\text{assigned}}$.

Eventually, the generated instance data are written to a text file and can be used for testing. The data are written to the text file in a standardized format such that they can be read in again easily.

To characterize an instance beyond the input parameters, a measure denoted by rRl_μ is calculated for the instance. This measure rRl_μ deals with the size of a skill requirement r_{pst} in relation to the availability of those workers that master skill s. In the following we refer to an instance of the project selection problem. In case of an instance of the workforce assignment problem, the definition is analog. For the measure rRl_μ, we consider each positive skill requirement $r_{pst} > 0$, $p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}}$, $s \in \mathcal{S}_p$, $t \in \mathcal{T}_p$. Let r^{num} denote the number of these positive skill requirements in an instance. For the availability of the workers, we take into account their skill levels, i.e., a worker k with $R_{kt} = 150$ and $l_{ks} = 2$ has an availability of 300 man-hours with respect to skill s in period t. The measure rRl_μ represents the average of the ratios of all positive skill requirements to the average availability across all suitable workers in the corresponding period, where the availability is adjusted for skill levels.

$$rRl_\mu := \frac{\sum_{p \in \mathcal{P}^{\text{ongoing}} \cup \mathcal{P}^{\text{must}} \cup \tilde{\mathcal{P}}} \sum_{s \in \mathcal{S}_p} \sum_{t \in \mathcal{T}_p} \frac{r_{pst}}{\frac{1}{|\mathcal{K}_s|} \sum_{k \in \mathcal{K}_s} R_{kt} l_{ks}}}{r^{\text{num}}} \tag{7.1}$$

If the average ratio rRl_μ is greater than 1, an average worker cannot fully accomplish an average skill requirement in a period even if he devotes all his time to this requirement. If rRl_μ is considerably lower than 1, an average worker can accomplish several skill requirements within a period. The average ratio rRl_μ depends on the values of several input parameters of the instance generator. It increases, for example, when, ceteris paribus, ρ^{proj} increases, the number of projects decreases, or the number of workers increases.

All test instances that we generated and used for the performance analyses which we present in the three following sections can be downloaded as text files from the internet website `http://www.wiwi.tu-clausthal.de/abteilungen/unternehmensforschung/forschung/benchmark-instances/`. Other researchers are invited to develop alternative solution methods for the three problems that we tackle and to use our instances for performance tests of their methods such that results are comparable.

7.2 Analysis for the project selection problem

For the numerical analysis of the project selection problem, we ran the MIP solver of CPLEX 12.4 on various sets of instances. We tested different solver settings, e.g., different search strategies. With respect to instances, we varied instance parameters such as the number of projects. One goal of parameter variation was to explore the impact of these parameters and to explore the limits of the instance size up to which solutions of acceptable quality can be found in acceptable time. Additionally, we investigated two concepts that enhance workforce flexibility, namely, multi-skilling and skill chaining. To examine the effect of multi-skilling, which is also called cross-training, we varied the number of skills mastered by the workers. Skill chaining requires a specific distribution of skills in the workforce and allows to accommodate shifts in skill requirements. To examine the advantage of skill chaining, we compared different skill distributions. For this comparison, we developed a new flexibility measure.

The following list gives an overview of the objects of the analysis.

- Solver settings: maximum number of threads, search strategy (best bound vs. depth-first), time limit
- Instance parameters: $\tilde{P}$, K, ρ^{proj}
- Flexibility concepts: multi-skilling, skill chaining

The starting point of our analysis is a test set of ten instances, which were constructed by the instance generator. To generate these ten instances, we set $K = 100$, $D = 10$, $\hat{P} = 150$, $S = 10$, and $T = 12$. Since we set the share of ongoing and of mandatory projects to 0, $\tilde{P} = \hat{P} = 150$ holds, i.e., a subset of 150 projects can be selected for the portfolio, while $|\mathcal{P}^{\text{ongoing}}| = |\mathcal{P}^{\text{must}}| = 0$. The members of each department share a common skill (*boolSameSkill* = true) and each worker $k \in \mathcal{K}$ masters between 1 and 3 skills (*boolNumSkillsRand* = true with $KminS = 1$ and $KmaxS = 3$). Each project requires between $PminS = 5$ and $PmaxS = 7$ skills. The input parameter ρ^{proj} was set to 2.5.

The data of an instance instantiated our MIP model (4.1)–(4.7) for project selection. The respective model was solved by the MIP solver of CPLEX; we selected the dual simplex method of CPLEX for solving LP relaxations within the branch-and-cut procedure.[5] For each instance, we set a time limit $t^{\max}$ of 300 seconds (5 minutes) and restricted the optimization process to 1 thread. For all other parameters of the MIP solver, we chose the default settings. The default settings imply that the best bound rule is applied for node selection. This rule prescribes that the node with the most promising bound is selected from those nodes that wait for processing during branch-and-cut, i.e., from those nodes of the branch-and-cut tree that have been generated but that have not been explored yet. The bound of a node is the objective function value of the solution to the LP relaxation which is associated with that node. Since the project selection problem is a maximization problem, a node with the highest bound is selected for processing, because the higher the bound of a node, the more promising is the node.

Results for each of the ten instances are given in Table 7.2. In the column headed "P", the number of selected projects is stated and the column "Objective" reports the objective

[5] See on page 111 in Section 6.1 why we chose the dual simplex method.

function value of the corresponding solution. The column "Solution status" indicates that a feasible solution was found for each instance, but only for eight instances the solver could prove optimality of the respective solution within the time limit. The solutions of instances 3 and 10 can be optimal, but need not. When the time limit was hit for these instances, there remained at least one unexplored node whose bound was greater than the best objective function value found so far, i.e., greater than 1785 and 1870, respectively. For instances 3 and 10, the column headed "*Gap*" reports the relative deviation of the best solution found from the best bound. This deviation is very small in both cases. For the other instances, the deviation is zero. A value of zero is indicated by a dash (–) in Table 7.2 and all following tables.

Table 7.2: Results for the instances of a test set with $\tilde{P} = 150$ projects ($K = 100$, $D = S = 10$, $t^{\max} = 300\,\text{s}$, 1 thread, search strategy: best bound)

Instance	Solution status	Time [s]	P	Objective	*Gap* [%]
1	optimal	144.0	103	1867	–
2	optimal	114.1	95	1876	–
3	feasible	300.0	102	1785	0.19
4	optimal	138.0	98	1847	–
5	optimal	200.5	97	1856	–
6	optimal	250.5	96	1744	–
7	optimal	210.2	104	1888	–
8	optimal	168.6	100	1870	–
9	optimal	24.2	105	1985	–
10	feasible	300.0	91	1870	0.34
Mean (μ)		185.0	99.1	1858.8	0.05

To calculate the relative deviation, let *BestBound* denote the value of the best bound at the time when the time limit is hit and let *ObjBestSol* denote the objective function value of the best integer-feasible solution that was found within the time limit. Then, our definition of the relative gap, which we denote by *Gap*, reads as follows.

$$Gap := \frac{|BestBound - ObjBestSol|}{BestBound} \cdot 100\,\%$$

If optimality is proven, *Gap* is set to 0.

The last row of Table 7.2 shows how we summarize results of a test set. For several quantities, we calculate the mean μ in order to represent the distribution of the quantity across the instances of a test set. For the mean, the results of all ten instances are considered, i.e., the mean is always calculated by averaging over all ten instances. Let x_i, $i = 1, \ldots, n$, denote a result for instance i. Then, the mean μ is defined as follows.

$$\mu := \frac{1}{n} \sum_{i=1}^{n} x_i$$

We use this aggregate measure in the following to present and compare results of different test sets.

For a first comparison, we varied the number $\tilde{P}$ of projects that are considered for the project portfolio. The number $\tilde{P} = \left|\tilde{\mathcal{P}}\right|$ determines the number of binary variables z_p, $p \in \tilde{\mathcal{P}}$, of an instance and is thus supposed to have a major impact on solution time and quality. To generate the instance sets that are listed in the left-most column of Table 7.3, we kept all input parameters of the instance generator constant except for $\hat{P} = \tilde{P}$. Consequently, the values of the parameters associated with workers and departments, e.g., R_{kt}, l_{ks}, and rd_{dt}, are the same in instance 2 of the test set $\tilde{P} = 25$ and in instance 2 of the test set $\tilde{P} = 50$, for example. Note that the amounts of the skill requirements r_{pst} of the first 25 projects differ in both instances, because they are adjusted to meet a required ratio ρ^{proj} of 2.5. The time limit of the solver was set to 5 minutes for every instance.

The results of this first comparison are presented in Table 7.3. The first column of this table denotes the number of projects $\tilde{P}$ that can be selected in every instance of the corresponding test set. The second column headed *opt.* states for how many out of the ten instances belonging to the respective test set a proven optimal solution could be determined within the time limit. In the third column, the average solution time $time_\mu$ per instance for the test set is reported. The average number of selected projects P_μ is given in the fourth column, the average objective function value obj_μ of the selected portfolio is listed in the fifth column. Finally, information about the average relative gap gap_μ of the solutions are summarized in the last column of the table.

Table 7.3: Results for test sets with different numbers of projects $\tilde{P}$, $t^{\max} = 300\,\text{s}$, 1 thread, search strategy: best bound ($K = 100$, $D = S = 10$)

$\tilde{P}$	*opt.*	$time_\mu$ [s]	P_μ	obj_μ	gap_μ [%]
25	10	2.7	13.3	244.0	–
50	10	27.6	30.0	558.8	–
100	9	100.1	64.8	1199.4	0.04
150	8	185.0	99.1	1858.8	0.05
200	1	290.6	131.8	2508.1	0.18
250	0	300.0	165.1	3156.7	0.20
500	0	300.0	325.1	6194.2	5.65
750	0	300.0	412.6	7926.7	20.14
1000	0	300.0	139.4	2699.3	80.02
1500	0	300.0	105.8	2047.6	90.01
2000	0	300.0	275.9	5293.3	80.01
2500	0	300.0	360.8	6780.5	80.01
3000	0	out of memory			

Table 7.3 shows for small-sized instances that the mean computation time sharply increases with an increase in the number of projects $\tilde{P}$. For all solutions of the instances that belong to the test set $\tilde{P} = 250$, optimality could not be proved. Though, gaps are very small, i.e., the solutions are either optimal or almost optimal. In the following, we term these solutions with gaps of less than 3 % *near-optimal*. For larger-sized instances, the solver cannot guarantee to find near-optimal solutions within the time limit $t^{\max}$. For one instance in the set $\tilde{P} = 500$, the relative gap amounts to 55 %, while the average gap of the remaining nine instances is equal to 0.2 % only. In the set $\tilde{P} = 1000$, for eight

instances the gap is equal to 100 %, because $\boldsymbol{z} = \mathbf{0}$ was the only feasible solution found, i.e., $\boldsymbol{z} = \mathbf{0}$ was the only leaf of the branch-and-cut tree reached within the time limit. For all instances of the set $\tilde{P} = 3000$, the required memory exceeded the available memory before the time limit was hit.[6] The memory shortage occurred after the LP relaxation of the root node had been solved, but before the root node was branched. The memory shortage forced CPLEX to abort the solution procedure.

Two settings of the solver have considerable impact on solution time and solution quality for instances of the project selection problem. These two settings are the number of threads available to the solver and the search strategy. When CPLEX was allowed to use up to the maximum number of threads, i.e., up to 8 threads, we obtained the results that are listed in Table 7.4. Solution times for small-sized instances decreased compared to the solution times we observed when only 1 thread could be used. Though, the use of several threads increases the memory demand and led to out-of-memory errors for instances with $\tilde{P} \geq 750$ projects.[7]

Table 7.4: Results for test sets with different numbers of projects $\tilde{P}$, $t^{\max} = 300\,\text{s}$, up to 8 threads, search strategy: best bound ($K = 100$, $D = S = 10$)

$\tilde{P}$	$opt.$	$time_\mu$ [s]	P_μ	obj_μ	gap_μ [%]
25	10	2.3	13.3	244.0	–
50	10	22.1	30.2	558.8	–
100	10	62.1	64.8	1199.5	–
150	9	139.7	99.1	1858.8	0.03
200	7	260.4	132.0	2508.6	0.05
250	1	299.3	165.1	3156.9	0.16
500	0	300.0	339.5	6548.2	0.22
750	0	out of memory			

Next, we compared the search strategies *best bound search* and *depth-first search*, using only 1 thread for both strategies. Outcomes were twofold. On the one hand, solution times for small-sized instances were considerably higher in case of the depth-first strategy (cf. Table 7.5), because the first integer-feasible solution $\boldsymbol{z} \neq \mathbf{0}$ that is found by a best bound search tends to be better than the first non-trivial feasible solution encountered by a depth-first search. The earlier such a solution is found and the better such a solution is, the larger parts of the branch-and-cut tree can be fathomed early on.

On the other hand, the depth-first search performed better than the best bound search for instances of larger size with $\tilde{P} \in \{500, 750\}$, because the depth-first search quickly reaches leafs of the branch-and-cut tree, whereas the best bound search can spend all

[6] When we ran the 64 bit version of CPLEX, which could exploit 32 GB RAM in our case, the memory limit was not hit for any instance of the test set $\tilde{P} = 3000$. Solution quality reached by CPLEX 64 bit for instances of smaller size was the same as the quality reached by CPLEX 32 bit.

[7] When we ran the 64 bit version of CPLEX, which could access 32 GB RAM, memory was sufficient even for instances of the test set $\tilde{P} = 3000$. Though, for test sets $\tilde{P} \geq 750$, CPLEX 64 bit returned the trivial solution $\boldsymbol{z} = \mathbf{0}$ for at least one instance of the respective test set. In case of $\tilde{P} = 3000$, $\boldsymbol{z} = \mathbf{0}$ was returned for all instances after 300 seconds. For $\tilde{P} = 500$, solution quality of CPLEX 64 bit and CPLEX 32 bit was almost identical.

Table 7.5: Results for test sets with different numbers of projects $\tilde{P}$, $t^{\max} = 300\,\text{s}$, 1 thread, search strategy: depth-first ($K = 100$, $D = S = 10$)

$\tilde{P}$	*opt.*	$time_\mu$ [s]	P_μ	obj_μ	gap_μ [%]
25	10	2.5	13.3	244.0	–
50	10	34.2	30.0	558.8	–
100	7	162.1	64.9	1199.2	0.29
150	2	256.7	99.1	1854.0	0.68
200	1	299.1	131.1	2502.4	0.55
250	0	300.0	165.1	3153.2	0.40
500	0	300.0	339.1	6539.5	0.36
750	0	300.0	514.0	9887.5	0.28
1000	0	300.0	477.2	8997.5	32.93
1500	0	300.0	60.0	950.7	95.34
2000	0	300.0	275.9	5293.3	80.01
2500	0	300.0	360.8	6780.5	80.01
3000	0	out of memory			

the time for processing nodes that lie between the root node and leaf nodes of the tree. For $\tilde{P} = 500$, the best bound search found near-optimal solutions for nine out of ten instances. For the remaining instance, a solution with a gap of 54.66 % was reached. Hence, the average gap amounts to 5.65 % (cf. Table 7.3). For $\tilde{P} = 750$, the best bound search reached near-optimal solutions for eight out of ten instances, but reached only the trivial leaf $\boldsymbol{z} = \mathbf{0}$ for the two remaining instances resulting in an overall average gap of 20.14 %. The depth-first strategy, in contrast, led to near-optimal solutions for all test instances of the sets $\tilde{P} = 500$ and $\tilde{P} = 750$, resulting in an average gap of only 0.36 % and 0.28 %, respectively (cf. Table 7.5).

For $\tilde{P} = 1000$, both search strategies cannot provide solutions of acceptable quality within the time limit. The best bound search returned $\boldsymbol{z} = \mathbf{0}$ for eight instances and found near-optimal solutions for only two instances resulting in an average gap of 80.02 %, whereas the depth-first search returns $\boldsymbol{z} = \mathbf{0}$ only for one instance and near-optimal solutions with gaps less than 0.3 % for six instances. For three instances, the gaps lie around 75 % resulting in an average gap of 32.92 %.

Given the time limit of 300 seconds and 1 thread only, acceptable solution quality was achieved for $\tilde{P} \leq 250$ in case of the best bound search and for $\tilde{P} \leq 750$ in case of the depth-first search. The available memory of 4 GB RAM could accommodate the memory demand of both search strategies in the first 300 seconds for instances with up to 2500 projects. Out-of-memory errors occurred for both strategies for all ten instances with $\tilde{P} = 3000$.

To see how solution quality improves if more time is granted, we increased the time limit for the best bound search with 1 thread from 300 seconds to 3600 seconds, i.e., from 5 minutes to 1 hour. The corresponding results for the test instances are summarized in Table 7.6.

Out-of-memory errors occurred for all instances of the test set $\tilde{P} = 3000$ within the first three minutes, as we had already observed from the results in Table 7.3. For nine instances

Table 7.6: Results for test sets with different numbers of projects $\tilde{P}$, $t^{\max} = 1\,\text{h}$, 1 thread, search strategy: best bound ($K = 100$, $D = S = 10$)

$\tilde{P}$	*opt.*	$time_\mu$ [s]	P_μ	obj_μ	gap_μ [%]
100	10	132.8	64.9	1199.5	–
150	10	284.9	99.1	1858.8	–
200	10	804.3	131.8	2508.6	–
250	8	1767.6	165.2	3158.0	0.02
500	1	3408.5	340.1	6555.5	0.08
750	0	3600.0	514.9	9905.3	0.09
1000	0	3600.0	689.4	13 283.3	0.08
1500	0	3600.0	1047.0	20 097.6	0.07
2000	0	3600.0	1364.4	25 803.2	4.52
2500	0	out of memory			
3000	0	out of memory			

of the test set $\tilde{P} = 2500$, the memory demand led to an out-of-memory error, because the branch-and-cut tree required too much memory.[8] The out-of-memory error occurred, on average, after 50 minutes when 348 nodes had been processed and 346 nodes waited for processing. When the out-of-memory error occurred, for five out of the nine instances only the trivial solution $\boldsymbol{z} = \mathbf{0}$ had been found, for two instances a poor integer-feasible solution with a large gap had been determined, and for the remaining two instances near-optimal integer-feasible solutions had been found.[9] These results indicate that it takes considerable amounts of memory and time to determine near-optimal solutions for instances of very large size.

Within 1 hour, an optimal solution was found and verified for each instance of the test sets with $\tilde{P} \leq 200$. Let us call solutions whose relative gap is at most 3 % *high-quality solutions*, i.e., proven optimal and near-optimal solutions are high-quality solutions. Such high-quality solutions were found for all instances with $\tilde{P} \leq 1500$. Since 1 hour solution time is deemed acceptable for a long-term planning problem at the strategic level of a firm, high-quality solutions were found for instances with $\tilde{P} \leq 1500$ in acceptable time. For $\tilde{P} = 2000$, gaps were negligibly small for all instances but for one.

A closer look at the branch-and-cut solution process reveals that the upper bound on the optimal objective function value is tight from the very beginning on. The upper bound, which is derived from the solution of the LP relaxation of the root node, is already close to the optimal objective function value. The tightness of the upper bound is documented in Figure 7.1(a) for a medium-sized instance with $\tilde{P} = 250$ and in Figure 7.1(b) for a large-sized instance with $\tilde{P} = 2000$. Both figures show how the respective value of the upper bound and the respective objective function value of the incumbent solution change over

[8] Whenever for at least one instance of a test set an out-of-memory error occurred, we report "out of memory" in our tables, although the available memory may have been sufficient for some instances of the test set.

[9] When we ran CPLEX 64 bit for $\tilde{P} = 2000$, $\tilde{P} = 2500$, and $\tilde{P} = 3000$, out-of-memory errors did not occur. Using 1 thread, average relative gaps after 1 hour amounted to 0.04 %, 37.97 %, and 34.57 %, respectively. Using up to 8 threads led to gaps of 10.04 %, 0.04 %, and 17.75 %, respectively.

time. The incumbent solution is at any point in time the best integer-feasible solution found until this point in time. For the medium- and the large-sized instance, the first integer-feasible solution $\boldsymbol{z} \neq \mathbf{0}$ that is encountered is already very close to the upper bound and is hence a near-optimal solution. The difference between both instances is the time required to spot this first non-trivial integer-feasible solution. The time required to determine such a solution tends to increase with instance size.

In a second experiment, we varied the number of workers K to examine the impact of this parameter on solution time and quality. Besides the number of projects $\tilde{P}$, the number of workers plays a central role for the project selection problem, as the parameter K influences model size. The higher the number of workers, the higher is the number of variables $\hat{y}_{kpst}$ and the higher is the number of Constraints (4.3), which ensure that the availability of every worker is observed in every period.

We found that an increase in the number of workers has a less severe impact on solution time and quality than an increase in the number of projects. Nevertheless, an increase in K has a substantial impact on solution time. The impact on solution quality is less intense.

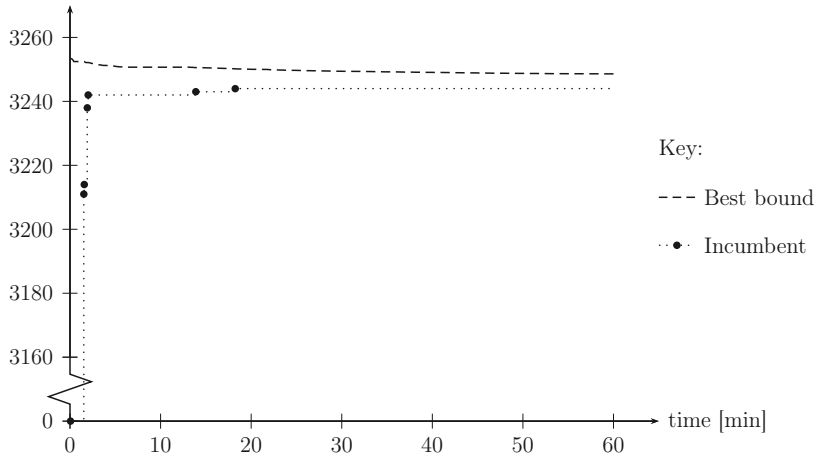
An increase in the number of workers raises the number of continuous variables $\hat{y}_{kpst}$, but does not increase the number of binary variables z_p. In general, binary variables have a larger impact on solution time, as they affect the size of the LP relaxation and, in particular, the number of nodes in the branch-and-cut tree, whereas the number of continuous variables only impacts the size of the LP relaxation that must be solved at each node. However, since the variable $\hat{y}_{kpst}$ is a four-index variable, the number of the continuous variables $\hat{y}_{kpst}$ is already large for modest numbers of workers, projects, skills, and periods. On the basis of three exemplary instances, Table 7.7 shows how the numbers of variables z_p, the numbers of variables $\hat{y}_{kpst}$, and the numbers of constraints change for the project selection model (4.1)–(4.7) when the number of projects is doubled and when the number of workers is doubled.

Table 7.7: Numbers of variables z_p, variables $\hat{y}_{kpst}$, and constraints for three instances of different size ($D = S = 10$, $T = 12$)

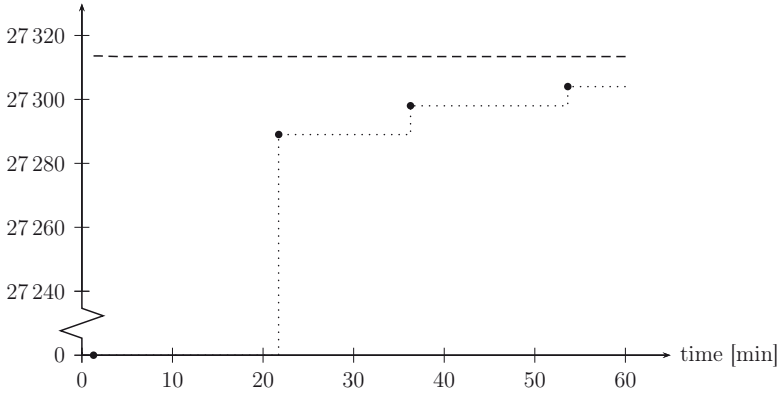
Instance	$\#z_{kp}$	$\#\hat{y}_{kpst}$	#constraints
$K = 100$, $\tilde{P} = 100$	100	39 674	3396
$K = 200$, $\tilde{P} = 100$	100	83 471	4596
$K = 100$, $\tilde{P} = 200$	200	81 958	5628

To examine the effect of the number of workers on solution time and quality, we conducted test series for two different time limits and for different numbers of projects. We considered 5 minutes and 1 hour as time limits. For each time limit, we carried out test series for three values of $\tilde{P}$. For each test series, we generated test sets with different numbers of workers (cf. Tables 7.8 and 7.9).

For a time limit of 5 minutes, instances with 250 workers and 50 projects as well as instances with 50 workers and 100 projects were solved to optimality (cf. Table 7.8). When there were 750 workers or more, only the trivial solution $\boldsymbol{z} = \mathbf{0}$ could be found for some instances with 75 and 100 projects.



(a) Instance 7 of the set $\tilde{P} = 250$ from Table 7.6 (absolute gap: 4, relative gap: 0.14 %)



(b) Instance 9 of the set $\tilde{P} = 2000$ from Table 7.6 (absolute gap: 7, relative gap: 0.03 %)

Figure 7.1: Value of the best bound and objective function value of the incumbent solution during branch-and-cut for a medium- and a large-sized instance

For a time limit of 1 hour, the number of high-quality solutions and the average relative gap are presented for each test set in Table 7.9. Solution quality decreases with the number of workers and the number of projects. For instances with 150 projects and $K \geq 1750$, the available memory was not sufficient to store the branch-and-cut tree.

Table 7.8: Average solution times $time_\mu$ [s], numbers of proven optimal solutions *opt.*, and average relative gaps gap_μ [%] for test sets with different numbers of workers K and projects $\tilde{P}$, $t^{\max} = 300\,\text{s}$ ($D = S = 10$, 1 thread, search strategy: best bound)

	$\tilde{P} = 50$			$\tilde{P} = 75$			$\tilde{P} = 100$		
K	$time_\mu$	*opt.*	gap_μ	$time_\mu$	*opt.*	gap_μ	$time_\mu$	*opt.*	gap_μ
25	2.6	10	–	4.7	10	–	25.1	10	–
50	4.6	10	–	15.0	10	–	39.8	10	–
100	16.3	10	–	62.7	10	–	117.7	9	0.04
250	72.9	10	–	166.5	8	0.19	225.9	5	0.36
500	190.0	8	0.48	272.1	3	2.03	300.0	0	2.40
750	254.9	5	1.29	300.0	0	31.01	300.0	0	50.97
1000	258.2	1	3.14	300.0	0	70.57	300.0	0	80.35

Table 7.9: Numbers of high-quality solutions and average relative gaps gap_μ [%] for test sets with different numbers of workers K and projects $\tilde{P}$, $t^{\max} = 1\,\text{h}$ ($D = S = 10$, 1 thread, search strategy: best bound)

	$\tilde{P} = 50$		$\tilde{P} = 100$		$\tilde{P} = 150$	
K	h.-q. sol.	gap_μ	h.-q. sol.	gap_μ	h.-q. sol.	gap_μ
100	10	–	10	–	10	–
250	10	–	10	–	10	–
500	10	–	10	0.11	10	0.10
750	10	–	10	0.22	10	0.27
1000	10	–	10	0.46	10	0.38
1250	10	–	10	0.88	8	10.66
1500	10	0.09	10	0.92	7	30.84
1750	10	0.31	10	1.79	out of memory	
2000	10	0.23	9	1.46	out of memory	

In another experiment, we varied the desired ratio of total project requirements to the corresponding availability of the workforce across test sets. For the generation of the test sets, we fixed K to 100 and $\hat{P}$ to 100. With $|\mathcal{P}^{\text{ongoing}}| = |\mathcal{P}^{\text{must}}| = 0$, i.e., with $sharePongoing := sharePmust := 0$, this led to $\tilde{P} = 100$ projects. All other input parameters of the instance generator were set to the same values as before. Only the parameter ρ^{proj} was varied by choosing $\rho^{\text{proj}} \in \{1, 2, \ldots, 14, 15, 20, 25, \ldots, 45, 50\}$. Hence, for example, the data of instance 2 of each test set are identical except for the the values of the positive project requirements r_{pst}. The greater the value of ρ^{proj}, the greater are the project requirements r_{pst} and the less projects can be selected, because workforce capacity was held constant.

The value of the parameter ρ^{proj} has a notable impact on solution time. Average solution times for the instances of all test sets are depicted in Figure 7.2 along with the

average number of projects that were selected in the proven optimal solutions. Since corresponding instances are identical across test sets except for the magnitude of project requirements, the respective optimization models exhibit identical numbers of variables and constraints. The instances differ only with respect to the number of feasible solutions and the number of top-quality solutions, whose objective function values are not far away from the optimal objective function value. The number of feasible portfolios decreases as ρ^{proj} increases, while the number of top-quality portfolios is expected to peak at that value of ρ^{proj} where an optimal portfolio contains around $\frac{\tilde{P}}{2}$ projects. The difference in the number of top-quality portfolios across test sets causes differences in solution time.

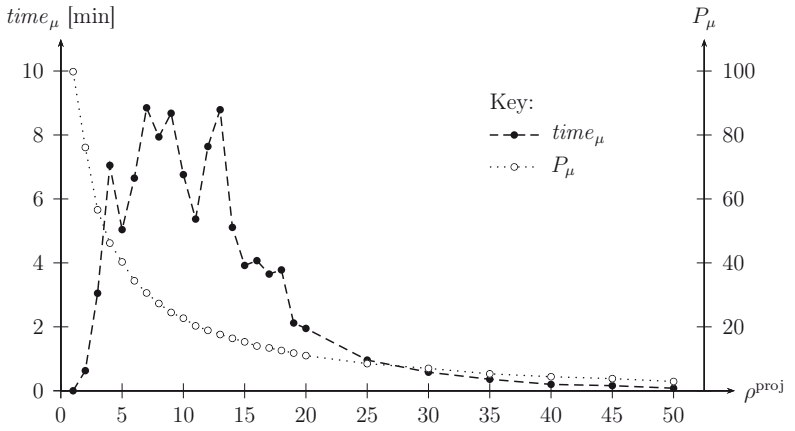


Figure 7.2: Average solution times $time_\mu$ and average numbers of selected projects P_μ for test sets with different ratios of project requirements to workforce availability ρ^{proj} ($\tilde{P} = 100$, $K = 100$, $D = S = 10$, $t^{\max} = 1\,\text{h}$, 1 thread, search strategy: best bound)

If the project requirements are so small that all projects can be selected as in the case of our test set $\rho^{\text{proj}} = 1$, the solution to the LP relaxation of the root node is integer-feasible and, thus, the optimal solution is found quickly. In this case, there is only one top-quality portfolio. If the project requirements are so large that all feasible portfolios contain at most one project, there are at most 100 top-quality solutions for an instance of our test sets. Among these solutions, an optimal one can be found in short time.

If the portfolio can contain up to $\frac{\tilde{P}}{2}$ projects, i.e., if a medium value was chosen for ρ^{proj}, then around $\binom{\tilde{P}}{\frac{\tilde{P}}{2}} = \frac{\tilde{P}!}{\frac{\tilde{P}}{2}!\,\frac{\tilde{P}}{2}!}$ top-quality solutions may exist, e.g., in our case with $\tilde{P} = 100$, there may exist approximately 10^{29} top-quality solutions. The branch-and-cut procedure has to process more nodes than for extremely small or large values of the parameter ρ^{proj}. For our test sets, on average 345 nodes had to be processed in case of $\rho^{\text{proj}} = 2$ per instance, 2113 nodes in case of $\rho^{\text{proj}} = 9$, and only 102 nodes in case

of $\rho^{\text{proj}} = 40$. Solution time peaked when an optimal portfolio comprised between 15 projects ($\rho^{\text{proj}} = 14$) and 50 projects ($\rho^{\text{proj}} = 4$).

In our last two experiments for the project selection problem, we considered two concepts that are related to workforce flexibility: multi-skilling, which is also called cross-training, and skill chaining. Skill chaining refers to a particular allocation of skills to workers that allows to shift workload from a highly utilized worker via a chain of workers to a worker that is not fully occupied. The concept of skill chaining will be explained in more detail below. Multi-skilling and skill chaining increase workforce flexibility. This flexibility facilitates a higher capacity utilization, i.e., it raises the capacity of the workforce without extending the working time limit that is agreed upon in labor contracts. The aim of our experiments was to determine the advantage of flexibility gained through multi-skilling and skill chaining. To this aim, we simulated workforces with different degrees of multi-skilling and we considered different skill configurations that represent different skill chaining strategies.

The scope of the advantage that both flexibility concepts provide is not obvious in our case. In regard to multi-skilling, there is no doubt about the existence of an advantage of a higher degree of multi-skilling in our case. A firm whose workforce is more flexible due to a high degree of multi-skilling can always select a project portfolio which is at least as beneficial as that portfolio that a firm with a less flexible workforce can select. The question is whether the advantage of multi-skilling is only marginal or so substantial that investments in cross-training are justifiable. In general, one might expect that an optimal portfolio exhibits a high level of utilization of the workforce. So, every worker may spend almost his complete time for project work. It is not obvious whether an additional skill for each worker makes it possible to select a portfolio that is considerably more beneficial. In regard to skill chaining, there is no doubt that longer chains provide more flexibility in the simple case where each worker masters two skills and the number of skills equals the number of workers. But again, the magnitude of the effect is not obvious. Moreover, in our problem setting the number of workers tends to exceed the number of skills by far. In this case, it is difficult to estimate the effects that different skill configurations will have.

In Section 2.2, we have reported on several studies that investigated the value of multi-skilling or resource flexibility and the value of chaining. The studies yielded consistent results for various problems: The marginal return of flexibility decreases and chaining tends to be advantageous. However, to the best of our knowledge, the value of multi-skilling has not been investigated for a project selection problem so far, and with respect to chaining, in most cases only simple chaining strategies have been considered. In the following, we will examine the value of multi-skilling and of chaining with respect to project selection. First, we look at the impact that different degrees of multi-skilling have. Then, we will compare different skill configurations that are characterized by the same degree of multi-skilling. Among the skill configurations, there is a simple and an advanced chaining strategy.

To assess the advantage of multi-skilling for project selection, we carried out four test series, one for each combination of two different instance sizes and two different parameter settings. With respect to instance size, we varied the number of workers and projects; with respect to parameter settings, the value of the parameter ρ^{proj} was varied (cf. Table 7.10). For each test series, we generated six test sets by setting $KminS := KmaxS$ with $KmaxS \in \{1, 2, \ldots, 6\}$, i.e., we generated one set of instances where each worker

masters exactly one skill, another set where each worker masters exactly two skills, and so on. For a test set with $KmaxS = 3$, $|\mathcal{S}_k| = 3$ holds for all $k \in \mathcal{K}$. From now on, we refer to such a test set as test set $|\mathcal{S}_k| = 3$. The instances of the test sets that belong to a series differ only in the number of skills mastered by the workers. For example, in an instance of the test set with $|\mathcal{S}_k| = 3$, every worker masters three skills, and in the corresponding instance of the test set with $|\mathcal{S}_k| = 4$, every worker masters four skills, namely, the same skills at the same levels as in the test set with $|\mathcal{S}_k| = 3$ plus an additional skill. All other data, e.g., the availabilities R_{kt} and the project requirements r_{pst}, are identical in these two instances and in all corresponding instances across the six test sets of a series.

These identical data are necessary to compare the different degrees of multi-skilling. To obtain these identical data across test sets, the instance generator was slightly modified. The modification affected the impact of parameter ρ^{proj} on the skill requirements r_{pst}. In the unmodified version, adjustments of skill requirements r_{pst} take into account the skill sets of the workers in order to achieve the desired ratio ρ^{proj} of project requirements to workforce availability. The modified version of the instance generator ignores the skill sets and returns identical requirements r_{pst} for any degree of multi-skilling, i.e., for different values of $|\mathcal{S}_k|$.

Our results confirm the finding of other researchers that marginal returns of flexibility decrease. The results for our test series are summarized in Table 7.10. We determined optimal solutions for all instances of all four test sets. First of all, note that solution time increases with increasing number of skills per worker. Increasing computation times are a disadvantage of greater flexibility and can even be a pitfall, as documented by Walter and Zimmermann (2012), who called the phenomenon of increasing computation times *the curse of flexibility*. Average objective function values obj_μ increase strictly monotonically with increasing flexibility of workers. Though, marginal benefits diminish when flexibility increases. The additional value of a fifth skill is almost negligible. For our test sets, limited flexibility, say three or four skills per worker, is almost as beneficial as maximum flexibility with six skills per worker.

The average total utilization $util_\mu$ of workers for the test sets from Table 7.10 is displayed in Figure 7.3. The utilization $util_\mu$ is the average utilization across the ten instances of a test set. It considers project work and departmental workload; that is why we call it *total* utilization. The mean total utilization *util* of a solution for a single instance is computed as the ratio of the total time spent for project and departmental work to the total availability:

$$util := \frac{\sum\limits_{k\in\mathcal{K}} \sum\limits_{p\in\hat{\mathcal{P}}_k^{\text{suit}}} \sum\limits_{s\in\mathcal{S}_{kp}^{\text{match}}} \sum\limits_{t\in\mathcal{T}_p} \hat{y}_{kpst} + \sum\limits_{d\in\mathcal{D}} \sum\limits_{t\in\mathcal{T}} rd_{dt}}{\sum\limits_{k\in\mathcal{K}} \sum\limits_{t\in\mathcal{T}} R_{kt}}$$

The mean utilization due to departmental requirements is about 18 % for each instance.

Figure 7.3 discloses that worker utilization is small for a mono-skilled workforce. Utilization increases sharply when one or two secondary skills are mastered by the workers. With two or three secondary skills, considerably better portfolios can be selected. These portfolios tend to contain more projects. The additional skills enable the workforce to cope with unbalanced skill requirements. Further skills do not help to select substantially better portfolios. Thus, utilization does not increase further. Note that the number of

Table 7.10: Average solution times $time_\mu$ [s], average numbers of selected projects P_μ, and average optimal objective function values obj_μ for test sets with different numbers of skills per worker $|\mathcal{S}_k|$ ($D = S = 10$)

	$K = 50,\ \tilde{P} = 50$						$K = 100,\ \tilde{P} = 100$					
	$\rho^{\text{proj}} = 10$			$\rho^{\text{proj}} = 20$			$\rho^{\text{proj}} = 10$			$\rho^{\text{proj}} = 20$		
$\lvert\mathcal{S}_k\rvert$	$time_\mu$	P_μ	obj_μ	$time_\mu$	P_μ	obj_μ	$time_\mu$	P_μ	obj_μ	$time_\mu$	P_μ	obj_μ
1	1.4	7.8	156.7	0.1	2.7	54.0	209.5	21.7	461.6	36.0	10.5	223.7
2	16.8	14.1	295.9	6.4	6.7	133.6	489.9	33.4	710.2	1118.4	19.7	427.5
3	16.4	16.7	346.4	30.4	9.2	185.1	559.6	35.7	753.7	1157.9	21.3	473.0
4	19.6	17.5	365.5	33.6	9.5	208.3	592.5	36.7	772.7	1584.0	22.6	491.7
5	21.8	17.4	372.8	34.5	10.3	218.7	792.5	37.3	783.3	1733.1	23.1	501.5
6	22.5	17.4	378.3	53.3	10.3	222.3	623.1	37.8	791.2	1788.6	23.3	509.0

selected projects P and the utilization *util* can decrease with additional skills, because the benefit b_p of a project $p \in \tilde{\mathcal{P}}$ is not proportional to project size. However, the optimal objective function value increases with increasing flexibility.

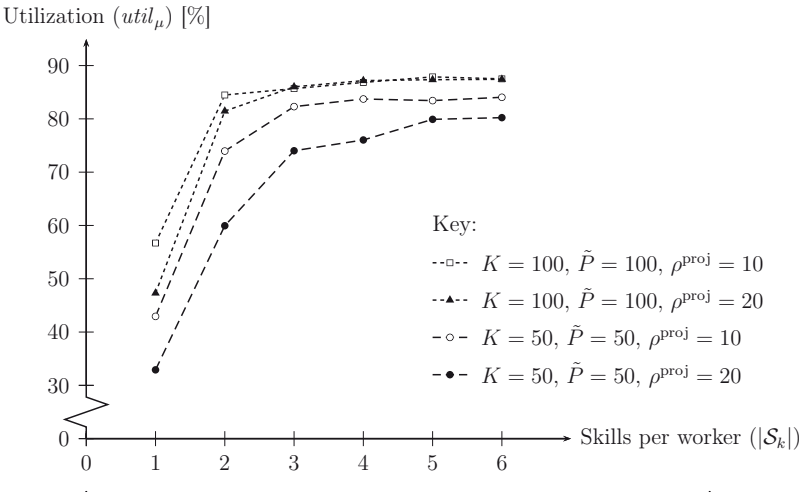


Figure 7.3: Average total utilization $util_\mu$ of a worker depending on the number of skills per worker $|\mathcal{S}_k|$ ($D = S = 10$, $\rho^{\text{dep}} = 0.2$)

To analyze the impact of chaining, we investigated three skill chaining strategies. The analysis comprised three approaches to assess the quality of the chaining strategies: We solved instances of the project selection problem, ran a simulation experiment, and applied simple flexibility measures. The three different skill chaining strategies are illustrated in Table 7.11 for an example with $K = 12$ workers, $D = 6$ departments, and $S = 6$ skills. For all strategies, $|\mathcal{S}_k| = 2$, $k \in \mathcal{K}$, holds and each worker $k \in \mathcal{K}$ masters a primary skill that depends on his department and a secondary skill. We assume that all skill levels l_{ks}, $k \in \mathcal{K}$, $s \in \mathcal{S}_k$, are equal to 1. Strategy or configuration A, called "short chains", features several chains, which involve two skills and two workers only. Configuration B, termed "long identical chains", contains long chains with an identical sequence of skills. Each chain involves all skills and all workers. This configuration is a classical, simple chaining strategy. Configuration C, called "diverse chains", is a representative for an advanced chaining strategy where the secondary skill of each worker is selected such that chains are created that feature more diverse skill sequences.

For the first approach, we solved instances of the project selection problem that represent the different skill chaining strategies. We prepared three test series and for each test series we generated one test set for each chaining strategy. The three test series differ in the parameters K, $\tilde{P}$, and ρ^{proj} (cf. Table 7.12). With respect to the number of skills per worker and skill levels, all instances follow the general pattern in Table 7.11, i.e., $|\mathcal{S}_k| = 2$, $k \in \mathcal{K}$, and $l_{ks} = 1$, $k \in \mathcal{K}$, $s \in \mathcal{S}_k$, holds in all instances.

Our computational results, which are outlined in Table 7.12, approve the advantage

Table 7.11: Skill configurations of three different skill chaining strategies

		$\mathcal{S}_k$, $k \in \mathcal{K}_d$					
		Config. A		Config. B		Config. C	
d	$\mathcal{K}_d$	(short chains)		(long identical chains)		(diverse chains)	
d_1	$\{k_1, k_2\}$	{1,2}	{1,2}	{1,2}	{1,2}	{1,2}	{1,3}
d_2	$\{k_3, k_4\}$	{1,2}	{1,2}	{2,3}	{2,3}	{2,3}	{2,4}
d_3	$\{k_5, k_6\}$	{3,4}	{3,4}	{3,4}	{3,4}	{3,4}	{3,6}
d_4	$\{k_7, k_8\}$	{3,4}	{3,4}	{4,5}	{4,5}	{4,5}	{4,1}
d_5	$\{k_9, k_{10}\}$	{5,6}	{5,6}	{5,6}	{5,6}	{5,6}	{5,2}
d_6	$\{k_{11}, k_{12}\}$	{5,6}	{5,6}	{6,1}	{6,1}	{6,1}	{6,5}

of longer chains compared to short identical chains and beyond that deepen the current knowledge about chaining strategies. For the three test series, the ranking of the three skill configurations or chaining strategies is consistent. With respect to solution quality, which is denoted by obj_μ, configuration C, which features diverse chains, performs best, followed by configuration B with long identical chains. Skill configuration A with its short chains leads to solutions of inferior quality in all test series. The following two examples, which refer to the sample data in Table 7.11, explain this ranking. The first example, Example 7.1, indicates that skill configurations B and C are superior to A. Example 7.2 explains the different outcomes for configurations B and C.

Example 7.1 Assume that the data in Table 7.11 represent the situation of a firm with $K = 12$ and $D = 6$ where the workforce masters $S = 6$ skills in total; let the firm consider the three skill configurations A, B, and C. Additionally, let $T = 1$, $R_{kt_1} = a > 0$, $k \in \mathcal{K}$, and $l_{ks} = 1$, $k \in \mathcal{K}$, $s \in \mathcal{S}_k$. Assume that the total requirement for each skill amounts to $2a$. For all skill configurations, let the two workers of department d_1 accomplish the requirement for skill 1, let the two workers of department d_2 do the workload of skill 2, and so forth. Each of the three skill configurations can satisfy all skill requirements.

Now, let there be a shift in skill requirements such that the requirement for skill 2 drops to 0, while the requirement for skill 4 doubles to $4a$. Skill configurations B and C can accommodate this shift, whereas configuration A cannot meet all requirements anymore.

In case of configuration A, the members of department d_3 can accomplish the additional requirements of skill 4, but then the requirement of skill 3 is not accomplished. At the same time, the workers in department d_2 are idle. In consequence, either workload of skill 3 or skill 4 or some workload of both skills cannot be allocated, because there is no chain that links workers who master skill 2 with workers mastering skill 4.

In case of skill configuration B, such chains exist. For example, the workers k_5 and k_6 of department d_3 can accomplish the additional requirement for skill 4. Their workload of skill 3 is shifted to the workers of department d_2 who would be idle otherwise. Put differently, the chains s_2–k_3–s_3–k_5–s_4 and s_2–k_4–s_3–k_6–s_4, for instance, can be exploited.

If skill configuration C is in force, the chain s_2–k_3–s_3–k_5–s_4 and worker k_4, i.e., the chain s_2–k_4–s_4, can be used to accommodate the shift in requirements. □

Table 7.12: Average solution times $time_\mu$ [s], average numbers of selected projects P_μ, average optimal objective function values obj_μ, and average utilization of workers $util_\mu$ [%] for three different skill chaining strategies ($D = S = 10$; $|\mathcal{S}_k| = 2$, $k \in \mathcal{K}$)

	$K = 50$, $\tilde{P} = 50$				$K = 100$, $\tilde{P} = 100$							
	$\rho^{\text{proj}} = 10$				$\rho^{\text{proj}} = 10$				$\rho^{\text{proj}} = 20$			
Chaining strategy	$time_\mu$	P_μ	obj_μ	$util_\mu$	$time_\mu$	P_μ	obj_μ	$util_\mu$	$time_\mu$	P_μ	obj_μ	$util_\mu$
Short chains (A)	5.3	6.8	137.2	49.4	602.4	18.7	400.2	65.3	133.3	9.1	203.1	55.9
Long identical chains (B)	14.1	8.9	182.2	63.2	653.0	21.9	477.7	77.5	512.2	12.0	257.2	71.1
Diverse chains (C)	13.5	9.8	208.3	72.5	562.4	23.1	504.8	81.2	619.9	13.0	289.1	75.3

The following example indicates that skill configuration C is, on average, superior to B.

Example 7.2 (continued from Example 7.1) Assume that the first shift in skill requirements persists and that a second shift occurs, where the requirement for skill 1 falls from $2a$ to 0 and the requirement for skill 6 doubles to $4a$.

In case of skill configuration B, each demand shift can be absorbed if it occurs on its own, as there are corresponding chains. However, both shifts cannot be handled if they occur simultaneously. If a chain is used to reallocate workload for one demand shift, every chain that is capable to absorb the other shift is blocked. Then, only one of the shifts can be fully accommodated. Alternatively, both shifts can be partially accommodated. If the first shift concerning skills 2 and 4 is accommodated as in Example 7.1, the chains from the workers who master skill 1 to those workers who master skill 6 are blocked. Likewise, exploiting a chain from skill 1 to skill 4 blocks a chain from skill 2 to skill 6.

Configuration C can accommodate one demand shift completely and the other partially at the same time. Assume that the first shift is accommodated as in Example 7.1. For the second shift in skill requirements, the chain s_1–k_2–s_3–k_6–s_6 from skill 1 to skill 6 can be exploited to adapt to the demand shift.

The advantage of configuration C compared to B is that the skill sequences in the chains of configuration C are more diverse. This diversity increases the probability that there is an unblocked chain that can accommodate a further shift of skill requirements. □

In a second approach to evaluating the skill configurations A, B, and C, we conducted a simulation. Our aim was to assess the skill configurations from Table 7.11 on a broader basis than in Example 7.2. We ran a simulation with randomly generated skill demands and recorded for each demand scenario the maximum demand which could be fulfilled by each skill configuration. The maximum was determined by solving a simple LP. For all 12 workers we set their availability to 1. This corresponds to $a = 1$ in Example 7.2. A demand scenario was generated as follows. For each of the six skills, a requirement value was randomly selected from the set $\{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4\}$. If the sum of all skill requirements, the total demand, exceeded 12, the total availability of the workforce, we repeatedly selected a skill with a positive requirement and reduced its requirement by 0.5 until a total demand of 12 was reached.

The simulation comprised 100 000 randomly generated demand scenarios. We did not exclude duplicate scenarios. The same demand scenarios were imposed on each skill configuration. The average total demand amounted to 10.732. The results of the simulation are summarized in Table 7.13, which confirms the ranking C, B, A that we expected from Example 7.2.

Note that there are even better skill configurations than configuration C. We randomly generated 1000 skill configurations, including duplicates, and tested each configuration against the 100 000 demand scenarios. The best configuration found could fulfill a demand of 10.585 on average. Like configuration C, the best configuration profits from chains with diverse skill sequences.

To assess the three skill configurations using a third approach, we applied a fast method that is based on simple flexibility measures. Simulation, which we used for the second approach, is one way to compare the quality of skill configurations; though, this way can be time-consuming. In Section 2.2, we introduced simple but well thought out measures

Table 7.13: Results of the simulation with 100 000 randomly generated skill demand scenarios for different skill configurations

	Average demand ...
... of all scenarios	10.732
... fulfilled by skill configuration A	9.348
... fulfilled by skill configuration B	10.302
... fulfilled by skill configuration C	10.541
... fulfilled by the best configuration found	10.585

proposed by Iravani et al. (2005) and Chou et al. (2011). We applied their measures to evaluate the three skill configurations A, B, and C from Table 7.11. The capacity and demand data required for the measures of Chou et al. were chosen in accordance with the data used for the simulation. This means that we assumed $c_k = 1$ for the capacity or availability of each worker $k \in \mathcal{K}$ and $E[\tilde{d}_s] = 1.789 \approx \frac{10.732}{6}$ for the expected demand for each of the six skills $s \in \mathcal{S}$. In the simulation, 10.732 was the average demand for all six skills per scenario. The values of the measures φ_{arc}, φ_{mean}, and $\varphi_{\text{eigenvalue}}$ devised by Iravani et al. (2005) and the values of the measures δ_i^{node} and $\delta_{ii'}^{\text{pairwise}}$ developed by Chou et al. (2011) are presented in Table 7.14.

Table 7.14: Values of different flexibility measures for the three skill configurations from Table 7.11, which represent different skill chaining strategies (a bracketed value for $\varphi_{\text{eigenvalue}}$ indicates that this value was computed—contrary to recommendation—for a disconnected graph)

	Skill configurations from Table 7.11		
Flexibility measure	Config. A (short chains)	Config. B (long identical chains)	Config. C (diverse chains)
φ_{arc}	24	24	24
φ_{mean}	1.33	4	4
$\varphi_{\text{eigenvalue}}$	(8)	24	24
$\varphi_{\text{mean}}^{\text{pairwise}}$	3.2	5.33	5.91
Lowest δ_i^{node}	2.24	2.24	2.24
Lowest $\delta_{ii'}^{\text{pairwise}}$	1.12	1.68	1.68
Second lowest $\delta_{ii'}^{\text{pairwise}}$	1.12	1.68	1.79

In addition, Table 7.14 contains a new flexibility measure, $\varphi_{\text{mean}}^{\text{pairwise}}$, that we developed based on the measure φ_{mean} of Iravani et al. (2005). We devised $\varphi_{\text{mean}}^{\text{pairwise}}$ because all the measures of Iravani et al. are indifferent with respect to skill configurations B and C. Their measures are indifferent because both configurations feature an identical structure flexibility matrix. The identity $M_B = M_C$ of the structure flexibility matrices reflects the reasoning in Example 7.1. When only the demand for one skill is increased and the demand for another one is decreased, capacity can be shifted to accommodate the imbalance

in case of both skill configurations. However, when the demand for two skills rises, while the demand for two other skills falls, skill configuration C outperforms configuration B as explained in Example 7.2. The flexibility measure $\varphi_{\text{mean}}^{\text{pairwise}}$ captures this performance difference. $\varphi_{\text{mean}}^{\text{pairwise}}$ is calculated similar to φ_{mean}. The latter requires to compute maximum flows between two single nodes, whereas $\varphi_{\text{mean}}^{\text{pairwise}}$ requires to determine maximal flows between two pairs of nodes.

The exact computation of $\varphi_{\text{mean}}^{\text{pairwise}}$ works as follows. For each quadruple $(s, s', \bar{s}, \bar{s}') \in \mathcal{S}^4$ with $s < s'$, $s < \bar{s}$, $s' \neq \bar{s}$, $\bar{s} < \bar{s}'$, and $s' \neq \bar{s}'$, we calculate the maximum number of non-overlapping paths that run from skill nodes s and s' through the bipartite graph G that represents the skill configuration to skill nodes $\bar{s}$ and $\bar{s}'$. The maximum number of non-overlapping paths is equal to the maximum flow that can be shipped from skill nodes s and s' to skill nodes $\bar{s}$ and $\bar{s}'$ when for all edges in G a maximum capacity of 1 is assumed. Given the maximum flow for all quadruples, we calculate the average number of paths per quadruple. $\varphi_{\text{mean}}^{\text{pairwise}}$ is defined as this average value.

When the measures φ_{mean} and $\varphi_{\text{eigenvalue}}$ are indifferent, the new measure $\varphi_{\text{mean}}^{\text{pairwise}}$ may be able to distinguish the flexibility of two skill configurations. The configuration for which $\varphi_{\text{mean}}^{\text{pairwise}}$ is larger, is the more flexible one. As can be seen from Table 7.14, $\varphi_{\text{mean}}^{\text{pairwise}}$ distinguishes skill configuration B from C and ranks the three configurations A, B, and C in the same order that resulted from the simulation experiment.

The set of measures provided by Chou et al. (2011) provides the same ranking. However, note that the measures of Chou et al. distinguish configuration B from C also only when pairs of nodes are considered. For all three configurations, the single node expansion ratio of each skill node equals 2.24 and the ratio of each worker node is equal to 3.58. In regard to configurations B and C, ties are not broken by the lowest pairwise expansion ratio but by the second lowest.

We can sum up the analysis of the three chaining strategies as follows. Results obtained from CPLEX for test instances, the simulation experiment, a refined version of a flexibility measure from Iravani et al. (2005), and the measures of Chou et al. (2011) confirm for the project selection problem that a skill configuration with diverse chains outperforms a configuration that follows the classical chaining strategy.

7.3 Analysis for the workforce assignment problem

In this section, we present the results of our numerical analysis for the workforce assignment problem. We compare our modeling approaches and analyze the performance of exact branch-and-cut methods provided by the solver package CPLEX in Subsection 7.3.1. In Subsection 7.3.2, the performance of our heuristic methods is analyzed. Before we look at the results, though, we will briefly show the potential of explicitly considering the number of assignments in order to emphasize the importance of our approach once more and we will give a short description of the test instances on which our analysis is founded.

Models that do not explicitly consider the number of assignments of workers to projects can provide solutions that imply a very large average team size, which may impede smooth project execution. To give an example for the potential of minimizing average team size, we compared the number of assignments for three models. The first model is model (4.1)–(4.7) for the project selection problem. In a solution of this model, the variables $\hat{y}_{kpst}$

implicitly constitute project teams for the selected projects. The second model is a linear program whose objective function minimizes the total working time. Since the time that must be spent for department work is given, this objective function can be written as Min. $\sum_{k\in\mathcal{K}} \sum_{p\in P_k^{\text{suit}}} \sum_{s\in\mathcal{S}_{kp}^{\text{match}}} \sum_{t\in\mathcal{T}_p} y_{kpst}$. The secod model can be seen as a model that minimizes variable costs which are incurred for every minute a worker is accomplishing project workload. Hence, the second model imitates some aspects of the model of Heimerl and Kolisch (2010a). The third model is model (4.19)–(4.29), i.e., the network model for the workforce assignment problem. This model explicitly minimizes the number of assignments.

To compare the outcomes of the three models, we first solved all instances of the test set $K = 50$, $\tilde{P} = 50$ from Table 7.8 on page 213 applying the first model, i.e., the model for project selection (cf. Table 7.15). From the solution of this model for each instance, we derived an instance for the other two models. Such a derived instance contained exactly those projects that were selected by the first model. The second column in Table 7.15 states for each instance the number of projects P that were selected by the first model. Hence, P projects were staffed by each of the three models, e.g., 29 projects in case of instance 1.

Table 7.15: Numbers of assignments of workers to projects for three different models, namely, maximization of portfolio benefit (project selection), minimization of variable costs, and minimization of the number of assignments (workforce assignment) (test set $K = 50$, $\tilde{P} = 50$ from Table 7.8 on page 213 with $D = S = 10$)

		Number of assignments		
Instance	P	Mod. project sel.	Mod. variable costs	Mod. workforce assignm.
1	29	551	521	217
2	30	619	592	239
3	29	527	520	213
4	31	547	549	227
5	27	533	490	202
6	32	625	606	273
7	27	482	439	196
8	31	620	586	249
9	26	510	457	180
10	32	598	559	224
Mean (μ)		561	532	222

Table 7.15 shows that the number of assignments can be drastically reduced if team size is considered explicitly. The solutions of the project selection problem feature an average project team size of about 19 workers. Average team size amounts to 18 workers for the second model, whereas average team size could be reduced to 7.5 workers for the third model.[10] Compared to the first and the second model, this is a reduction by

[10]In case of the third model, the time limit was set to 2 hours for each instance. The average gap of the

around 60 %. In the first and the second model, requirements of each skill $s \in \mathcal{S}$ are primarily allocated to those workers that master skill s with the greatest skill level. Such an allocation leads to scattering the workers across projects.

Having realized that it is necessary to explicitly consider the number of assignments in order to obtain small and efficient project teams, we will describe the instances that we generated to test our solution methods for the workforce assignment problem.

The basis of our performance analysis are three collections of test sets. We call these three collections testbed 1, testbed 2, and testbed 3. These three testbeds were used for both exact and heuristic approaches. Every test set of each testbed comprises 10 instances. As before, for all instances of a test set, the values of the input parameters of the instance generator were kept constant.

Testbed 1 comprises six test sets (cf. Table 7.16 on page 227, for example). The instances of these test sets range from small- to medium-sized instances. The numbers of workers K and projects P range from 10 to 100 each, the numbers of departments D and skills S are varied between 3 and 10. For all other parameters that are required by the instance generator, we used the default values.

Testbed 2 comprises instances of medium and large size (cf. Table 7.28 on page 243, for example). For testbed 2, we generated six test series. Across the test series, the number of workers K ranges from 40 to 1500, the number of projects is varied between 40 and 500, and the number of departments D and skills S lies between 6 and 75. Each test series is defined by a combination of the number of workers K, the number of projects P, the number of departments D, and the number of skills S. For each combination, i.e., for each test series, we chose appropriate values for the parameters $PminS$ and $PmaxS$ which define the range for the number of skills that are required by each project. We created three test sets for each test series. For the first set of instances, we used our default values for the remaining input parameters of the instance generator. For the second set, we increased the parameters $KminS$ and $KmaxS$, i.e., we changed the range from which the number of skills $|\mathcal{S}_k|$ for each worker k is selected. The second set is thus characterized by a more flexible workforce. For the third set of test instances, we changed the value of ρ^{proj} from 0.6 to 0.8 leading to a higher utilization of the workforce. In total, 18 test sets were generated, three for each of the six test series of testbed 2.

Testbed 3 was designed to test the impact of the parameter ρ^{proj}, which influences the ratio of total project requirements to the corresponding availability of the workforce. As in the case of the project selection problem (cf. Figure 7.2 on page 214), we found that this parameter has a notable impact on solution time. However, the relationship between ρ^{proj} and solution time is different from the relationship that we observed for the project selection problem. To generate test sets, we fixed K to 15, P to 15, and S to 4 (cf. Figure 7.6 on page 241, for example). For each $\rho^{\text{proj}} \in \{0.05, 0.1, 0.15, \ldots, 0.85\}$, a test set was created. In total, these are 17 test sets in testbed 3. From $\rho^{\text{proj}} = 0.9$ on, some instances of the respective test set are infeasible, because workforce availability is not sufficient to cover project and departmental workloads. In every instance, each worker masters at least one skill and at most three skills, while each project requires man-hours for all four skills. As in the case of the project selection problem, the corresponding

solutions to all ten instances in Table 7.15 amounts to 18.2 %; the minimum gap amounts to 13.2 %. For the other models, optimality of the solutions was proved fast.

instances of two arbitrary test sets are identical except for project requirements r_{pst}, which monotonically increase with ρ^{proj}.

In the following, we will report on the results that we obtained when testing the solution methods on the instances of these three testbeds.

7.3.1 Analysis of exact methods and of their support

In this subsection, we analyze alternative approaches to model the workforce assignment problem as a MIP. Furthermore, we analyze measures to support the solution process, which is delegated to a branch-and-cut solver. All those approaches and measures were described in Subsections 4.3.1 and 6.1.1. To be more precise, we test the various sets of big-M constraints, compare the standard to the network model, and check the effectiveness of the globally and locally valid inequalities that are based on lower bounds on the number of assignments of workers to single projects. Besides, we also examine the impact of solver and parameter settings. Our criteria are solution time and quality. Our goal of the analysis is to derive specifications of the MIP model and the solver that facilitate a solution process which is as efficient as possible. Additionally, we explore boundaries of the instance size up to which solutions of acceptable quality can be found in reasonable time. Two important results shall be mentioned in advance: First, instances of the workforce assignment problem cannot be solved as efficiently as instances of the project selection problem. Second, a high level of workforce utilization, which is a property that is common for real-life instances, makes instances particularly hard.

The following list summarizes which alternative model components and which solver and parameter settings are analyzed in which order in this subsection.

- Different sets of big-M constraints
- MIP formulations: standard model vs. network model
- Solver settings: search strategy (best bound vs. depth-first), time limit
- Globally and locally valid inequalities
- Instance parameters: ρ^{proj}, instance size (K, D, P, S)

In Subsection 4.3.1, we outlined three alternative sets of big-M constraints for the standard model, namely, Constraint sets (4.11), (4.17), and (4.18). Analogous constraint sets can be applied to the network model. With Example 4.2 on page 60, we demonstrated that Constraints (4.11) are tighter than (4.17), which in turn are tighter than (4.18). In Subsection 6.1.1, we presented tighter versions of the big-M constraints (4.11) and (4.17), namely, Constraints (6.5) and (6.6), respectively. These tighter versions do not only consider the availability of a worker, but also the project requirements and the departmental workloads. A universal ranking of (6.5) and (6.6) with respect to tightness is not possible. Depending on the instance, Constraint set (6.5) can be tighter than (6.6) or vice versa. Hence, it may be beneficial to include both constraint sets in the workforce assignment model. The inclusion of both sets will be denoted by "(6.5)+(6.6)" in the following.

To get a deeper insight into the tightness of the different big-M constraint sets and into their impact on solution time and quality, we tested the constraint sets including

the case (6.5)+(6.6) for the test sets of testbed 1. For each test set of testbed 1, we conducted three computational tests. In the first test, we solved the LP relaxation of the respective standard model using the dual simplex optimizer of CPLEX (cf. Table 7.16). In the second test, we solved the respective MIP model with a time limit of 5 minutes (cf. Table 7.17). For the third test, the time limit was increased to 1 hour (cf. Table 7.18).

Table 7.16: Average optimal objective function values obj_μ and average solution times $time_\mu$ in brackets for the LP relaxation of the standard model with different big-M constraints for testbed 1 ($\rho^{\text{proj}} = 0.6$, 1 thread)

Instance set			Standard model, LP relaxation					
			obj_μ ($time_\mu$ [s]) for big-M constraints ...					
K	P	$D = S$	... (4.11)	(4.17)	(4.18)	(6.5)	(6.6)	(6.5)+(6.6)
10	10	3	12.0 (0.01)	9.2 (0.01)	6.5 (0.01)	14.3 (0.01)	13.3 (0.02)	15.3 (0.02)
10	20	3	13.2 (0.01)	9.7 (0.03)	6.3 (0.01)	22.2 (0.08)	23.0 (0.12)	24.5 (0.15)
20	10	3	24.4 (0.03)	18.4 (0.05)	12.3 (0.01)	25.0 (0.03)	19.9 (0.08)	25.7 (0.06)
20	20	5	26.0 (0.06)	20.6 (0.10)	12.8 (0.02)	35.2 (0.15)	37.3 (0.32)	40.2 (0.38)
50	100	10	61.7 (1.11)	45.5 (3.53)	30.9 (0.27)	209.1 (7.27)	265.4 (34.95)	267.0 (37.01)
100	50	10	122.0 (5.63)	91.5 (15.75)	64.0 (0.32)	145.1 (8.68)	145.8 (27.69)	167.7 (33.06)

The results displayed in Table 7.16 confirm the tightness ranking for (4.11), (4.17), and (4.18). The big-M constraints (6.5) and (6.6) dominate (4.11) and (4.17), respectively. Constraints (4.18) are dominated by all other constraint sets with respect to tightness. Constraints (6.5) tend to be tighter than (6.6) for greater ratios of $\frac{K}{P}$, i.e., for greater values of the measure rRl_μ, which was defined by Equation (7.1) on page 204. For smaller ratios of $\frac{K}{P}$, (6.6) tends to be tighter than (6.5), because (6.6) is more efficient when project requirements are small compared to the availability of workers. The combination (6.5)+(6.6) results always in tighter bounds than (6.5) and (6.6) at the cost of longer solution times. While solutions times are moderate for the Constraint sets (4.11) and its tightened version (6.5), they are quite large for (4.17), its tightened version (6.6), and (6.5)+(6.6).

To see how the results for the LP relaxation are reflected in case of the MIP with a small time limit, consider Table 7.17. For some instances with up to 20 workers and 20 projects, proven optimal solutions were found with all big-M constraint sets except for the Constraint set (4.18). For small-sized instances, the combination (6.5)+(6.6) worked best, especially in regard to the average relative gap gap_μ and the average solution time $time_\mu$. For the test sets with 50 and 100 workers, i.e., for instances of medium size,

gaps are enormous for all tested big-M constraint sets and average objective function values differ widely across these test sets.

Table 7.17: Average optimal objective function values obj_μ, average relative gaps gap_μ, and average solution times $time_\mu$ for the standard model with different big-M constraints for testbed 1 ($\rho^{\text{proj}} = 0.6$, 1 thread, search strategy: best bound)

Instance set			Standard model, $t^{\max} = 300\,\text{s}$					
			obj_μ $\langle gap_\mu\ [\%]\rangle$ ($time_\mu$ [s]) for big-M constraints ...					
K	P	$D = S$	... (4.11)	(4.17)	(4.18)	(6.5)	(6.6)	(6.5)+(6.6)
10	10	3	18.5	18.5	18.5	18.5	18.5	18.5
			⟨–⟩	⟨0.9⟩	⟨1.3⟩	⟨–⟩	⟨1.5⟩	⟨–⟩
			(2.1)	(36.2)	(33.5)	(2.0)	(32.4)	(1.7)
10	20	3	28.5	28.6	28.7	28.5	28.7	28.5
			⟨0.3⟩	⟨4.9⟩	⟨6.0⟩	⟨0.3⟩	⟨5.4⟩	⟨0.3⟩
			(65.7)	(189.4)	(200.9)	(65.7)	(189.9)	(60.3)
20	10	3	30.3	30.4	30.4	30.3	30.4	30.3
			⟨2.9⟩	⟨9.2⟩	⟨10.4⟩	⟨2.9⟩	⟨9.0⟩	⟨1.2⟩
			(125.9)	(254.4)	(291.3)	(125.9)	(249.3)	(89.0)
20	20	5	47.7	47.4	48.1	47.8	47.9	47.5
			⟨3.1⟩	⟨6.2⟩	⟨11.0⟩	⟨3.3⟩	⟨7.3⟩	⟨2.8⟩
			(204.2)	(258.0)	(300.0)	(204.1)	(251.3)	(164.9)
50	100	10	730.9	505.3	515.9	730.9	*	575.5
			⟨166.8⟩	⟨84.3⟩	⟨93.1⟩	⟨166.8⟩	*	⟨109.6⟩
			(300.0)	(300.0)	(300.0)	(300.0)	*	(300.0)
100	50	10	496.2	598.3	397.6	496.2	583.2	519.9
			⟨185.1⟩	⟨255.6⟩	⟨150.8⟩	⟨185.1⟩	⟨246.7⟩	⟨199.3⟩
			(300.0)	(300.0)	(300.0)	(300.0)	(300.0)	(300.0)

*For one out of the ten instances, no integer-feasible solution was found within the time limit.

This fluctuation in average objective function values for medium-sized instances disappears when more computation time is granted. The results for a time limit of 1 hour are given in Table 7.18. For some small-sized instances, the branch-and-cut tree quickly grows beyond the memory limit of 4 GB, whereas the memory limit is not hit within 1 hour for larger-sized instances. However, in case of larger-sized instances, the tree size exceeds memory limits after longer computation times as well.

From our results for different big-M constraint sets for the standard model, we conclude that the combined Constraint set (6.5)+(6.6) is the best choice. The combination (6.5)+(6.6) provides the tightest LP relaxation and leads to the shortest average computation times for the MIP model. If the time limit is hit in case of larger-sized in-

Table 7.18: Average objective function values obj_μ, average relative gaps gap_μ, and average solution times $time_\mu$ for the standard model with different big-M constraints for testbed 1 ($\rho^{\text{proj}} = 0.6$, 1 thread, search strategy: best bound)

			Standard model, $t^{\max} = 1\,\text{h}$					
Instance set			obj_μ $\langle gap_\mu\ [\%] \rangle$ $(time_\mu\ [\text{s}])$ for big-M constraints ...					
K	P	$D = S$	... (4.11)	(4.17)	(4.18)	(6.5)	(6.6)	(6.5)+(6.6)
10	10	3	18.5	18.5	18.5	18.5	18.5	18.5
			⟨–⟩	⟨–⟩	⟨–⟩	⟨–⟩	⟨–⟩	⟨–⟩
			(2.1)	(50.7)	(91.8)	(2.1)	(230.3)	(1.7)
10	20	3	28.5	**	**	28.5	**	28.5
			⟨–⟩	**	**	⟨–⟩	**	⟨–⟩
			(94.6)	**	**	(94.8)	**	(64.9)
20	10	3	**	**	**	**	**	**
			**	**	**	**	**	**
			**	**	**	**	**	**
20	20	5	47.2	**	47.3	47.2	**	47.2
			⟨0.7⟩	**	⟨6.2⟩	⟨0.7⟩	**	⟨1.5⟩
			(1311.0)	**	(3101.4)	(1311.9)	**	(1484.9)
50	100	10	300.9	299.4	301.7	300.9	298.7	298.8
			⟨8.9⟩	⟨7.9⟩	⟨10.5⟩	⟨8.9⟩	⟨7.7⟩	⟨7.6⟩
			(3600.0)	(3600.0)	(3600.0)	(3600.0)	(3600.0)	(3600.0)
100	50	10	221.0	225.6	266.1	221	227.6	214.7
			⟨24.2⟩	⟨32.3⟩	⟨57.3⟩	⟨24.2⟩	⟨33.4⟩	⟨21.6⟩
			(3600.0)	(3600.0)	(3600.0)	(3600.0)	(3600.0)	(3600.0)

**For at least one out of the ten instances, an out-of-memory error occurred.

stances, big-M Constraint set (6.5)+(6.6) is competitive with respect to objective function values and gap values. In the following, we use this combined big-M constraint set for both the standard and the network model.[11]

When comparing the standard with the network model, the standard model turned out to be superior on average. For both models, we considered the same test sets as before and solved the corresponding LP relaxation (cf. Table 7.19) and the respective MIP with a time limit of 5 minutes and 1 hour (cf. Tables 7.20 and 7.21, respectively).

Table 7.19 reflects that the feasible region of the LP relaxation is identical in regard to $\boldsymbol{x}$ and $\boldsymbol{y}$ for both models. For most of the test sets, however, it takes more time to determine an optimal solution to the LP relaxation in case of the network model than in case of the standard model.

[11] For the network model, the left-hand side of Constraint set (6.5), i.e., the sum $\sum_{s \in \mathcal{S}_{kp}^{\text{match}}} y_{kpst}$, can be replaced by the flow variable f_{kpt}^{proj}.

Table 7.19: Average solution times $time_\mu$ [s] and average optimal objective function values obj_μ for the LP relaxation of the standard model and the network model for testbed 1 ($\rho^{\text{proj}} = 0.6$, 1 thread)

Instance set			LP relaxation			
			Standard model		Network model	
K	P	$D = S$	$time_\mu$	obj_μ	$time_\mu$	obj_μ
10	10	3	0.02	15.3	0.02	15.3
10	20	3	0.15	24.5	0.16	24.5
20	10	3	0.06	25.7	0.08	25.7
20	20	5	0.38	40.2	0.35	40.2
50	100	10	37.01	267.0	52.65	267.0
100	50	10	33.06	167.7	36.16	167.7

When we compare both MIP models for a time limit of 5 minutes, we considered only the small-sized instances, because it became obvious from the results displayed in Table 7.17 that a time limit of 5 minutes is not sufficient to reach an acceptable solution quality for the test sets $K = 50$ and $K = 100$. Table 7.20 shows that it is more time-consuming to solve the network model than to solve the standard model and that the network model performs worse than the standard model with respect to the number of proven optimally solved instances and the average relative gap.

Table 7.20: Average solution times $time_\mu$ [s], numbers of proven optimal solutions $opt.$, and average relative gaps gap_μ [%] for the standard model and the network model for the small-sized instances of testbed 1 ($\rho^{\text{proj}} = 0.6$, 1 thread, search strategy: best bound)

Instance set			$t^{\max} = 300\,\text{s}$					
			Standard model			Network model		
K	P	$D = S$	$time_\mu$	$opt.$	gap_μ	$time_\mu$	$opt.$	gap_μ
10	10	3	1.7	10	–	2.0	10	–
10	20	3	60.3	9	0.31	85.2	9	0.31
20	10	3	89.0	8	1.22	127.1	6	2.54
20	20	5	164.9	6	2.78	185.7	5	3.70

Also for a time limit of 1 hour, the standard model outperforms the network model on average (cf. Table 7.21). Note that the network model found better solutions to some instances than the standard model within the time limit. In case of the standard model, the memory limit of 4 GB was hit for one instance, leading to an out-of-memory error. In case of the network model, the memory limit was hit for three instances. When we applied a depth-first search to both models, the memory demand shrunk and the limit was not hit for any instance (cf. Table 7.21).

The memory demand of the depth-first search strategy was considerably less than the

Table 7.21: Average solution times $time_\mu$ [s], numbers of proven optimal solutions *opt.*, and average relative gaps gap_μ [%] for the standard model and the network model for testbed 1 ($\rho^{\text{proj}} = 0.6$, 1 thread)

			$t^{\max} = 1\,\text{h}$											
			Standard model						Network model					
Instance set			best bound			depth-first			best bound			depth-first		
K	P	$D = S$	$time_\mu$	*opt.*	gap_μ	$time_\mu$	*opt.*	gap_μ	$time_\mu$	*opt.*	gap_μ	$time_\mu$	*opt.*	gap_μ
10	10	3	1.7	10	–	2.0	10	–	2.0	10	–	2.0	10	–
10	20	3	64.9	10	–	56.0	10	–	94.4	10	–	91.0	10	–
20	10	3	**	**	**	1493.8	7	3.35	**	**	**	1228.5	7	3.38
20	20	5	1484.9	6	1.54	954.3	8	1.54	**	**	**	1262.8	7	2.28
50	100	10	3600.0	0	7.58	3600.0	0	7.93	3600.0	0	8.46	3600.0	0	8.95
100	50	10	3600.0	0	21.63	3600.0	0	25.84	3600.0	0	22.75	3600.0	0	25.12

**For at least one out of the ten instances, an out-of-memory error occurred.

demand of the best bound strategy, e.g., for some instances of the test set $K = 20$, $P = 10$, less than 1 MB instead of more than 1 GB RAM was required. Though, especially for the larger-sized instances, solution quality of the depth-first search strategy was worse than that provided by the best bound search strategy. From these and other experiments we concluded to stick to the best bound search strategy. When we wanted to determine proven optimal solutions for subsequent analyses and suffered from memory shortage, we stuck to the best bound strategy and transferred excessive tree data to the hard disc. The depth-first search was considered the last resort.

The computational advantages of the standard model compared to the network model can be attributed to differences in model size. The formulation of the network model requires more continuous variables and more constraints than the formulation of the standard model, as can be seen from the statistics for three exemplary instances in Table 7.22. Additional continuous variables of the network model are the flow variables f_{kpt}^{proj} and f_{kt}^{dep}. Additional constraints are the flow conservation constraints (4.21), which link the variables f_{kpt}^{proj} and y_{kpst}, $s \in \mathcal{S}_{kp}^{\text{match}}$. These additional model components are a computational burden for the simplex method, which is used to solve the LP relaxations during branch-and-cut.

Table 7.22: Numbers of binary variables x_{kp}, continuous variables, and constraints for three instances of different size ($D = S = 3$, $T = 12$)

Instance		Standard model			Network model		
K	P	$\#x_{kp}$	#cont. var.	#constr.	$\#x_{kp}$	#cont. var.	#constr.
10	10	97	632	1446	97	1184	1730
10	20	196	1250	2722	196	2250	3329
20	10	193	1040	2404	193	1998	2823

Both the standard and the network model suffer from quasi-symmetry of solutions and from a relatively weak LP relaxation. These unfavorable characteristics lead to long computation times for the corresponding MIP model. Quasi-symmetry means that there are many integer-feasible solutions which have an identical objective function value and that among these solutions of identical quality, many solutions resemble each other. The only difference between two solutions may be that in the first solution worker k_1 is assigned to project p_1 and k_2 to p_2, for example, while these assignments of k_1 and k_2 are swapped in the second solution. These two solutions are quasi-symmetrical.[12] If the first solution cannot be fathomed when a new best solution, i.e., a new incumbent solution, is found, neither can the second. Hence, many solutions whose objective function values are identical or almost identical must be enumerated. Unfortunately, there is no apparent answer to the problem of quasi-symmetry if it is desired to determine a proven optimal solution.[13] If optimality is not demanded, the negative consequences of quasi-symmetry

[12]The term *quasi-symmetrical* is used instead of the term *symmetrical*, because, in general, workers are not identical, especially with respect to their availability values. Nevertheless, quasi-symmetry and symmetry have the same detrimental effects on the performance of a branch-and-cut solver. Bosch and Trick (2005, pp. 83–85), for example, portray these effects for the case of symmetry.

[13]Bosch and Trick (2005, pp. 83–85) describe some ways to remedy the problem of symmetry to a certain extent. Though, those remedies cannot be readily transferred to the case of quasi-symmetry.

might be mitigated by specifying a maximum gap which must not be exceeded by a solution to be considered acceptable. However, this remedy requires a relatively tight lower bound, i.e., a tight LP relaxation in our case.

The LP relaxation of the workforce assignment models is not as tight as the LP relaxation of the project selection model. The relative weakness of the LP relaxation of the workforce assignment problem can be realized when comparing the results for the small-sized instances in Tables 7.16 and 7.18. The former table reports results for the LP relaxation of the standard model, the latter for the corresponding MIP model given a time limit of 1 hour. For an example, consider the instance set $K = 10$, $P = 20$ in conjunction with the combined big-M constraint set (6.5)+(6.6). The average lower bound derived from the LP relaxation is equal to 24.5, whereas the average optimal number of assignments amounts to 28.5: a deviation of 16.3 %. Compare this relative deviation to the deviations for the test sets $K = 25$, $\tilde{P} = 100$ and $K = 50$, $\tilde{P} = 100$ of the project selection problem (cf. Table 7.8 on page 213). Here, the deviations equal only 1.7 % and 1.2 %, respectively.

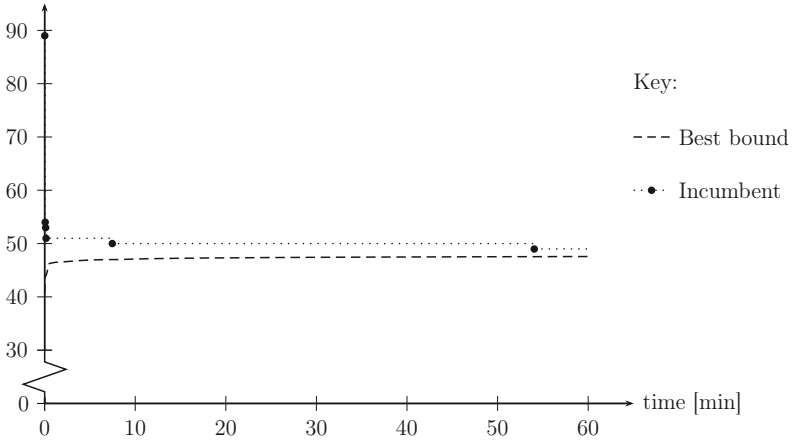
Additionally, for the workforce assignment problem, the first integer-feasible solutions encountered during branch-and-cut are far off the lower bound that is provided by the LP relaxation. This considerable gap can be seen in Figure 7.4, where the branch-and-cut process is documented for a small-sized instance with $K = 20$, $P = 20$ in Figure 7.4(a) and for a medium-sized instance with $K = 100$, $P = 50$ in Figure 7.4(b). Both figures show how the respective value of the lower bound and the respective objective function value of the incumbent solution change over time. Recall that the incumbent solution is at any point in time the best integer-feasible solution found thus far.

Valid inequalities can be a remedy against a weak LP relaxation. We outlined valid inequalities in Subsection 6.1.1. The purpose of valid inequalities is to tighten the LP relaxation. In a first step, we analyze globally valid inequalities (6.1) and (6.3), before we turn to locally valid inequalities (6.2) and (6.4).

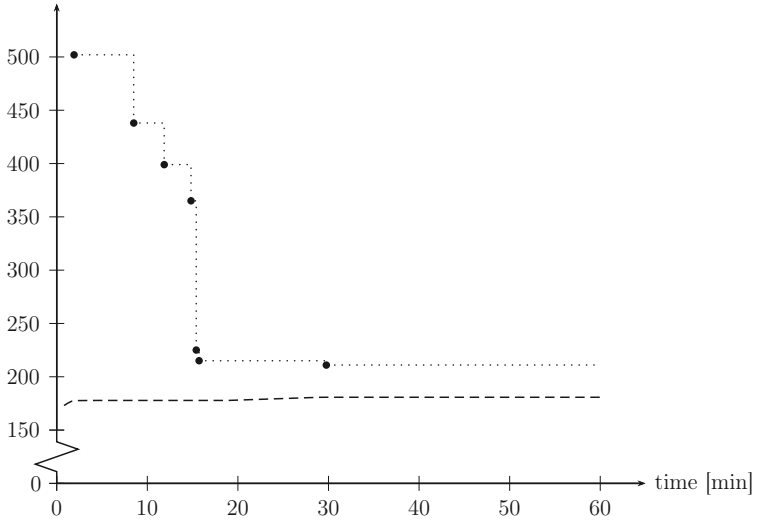
Table 7.23 shows how globally valid inequalities (6.1) and (6.3) affected the solution of the LP relaxation of the standard model. Our point of reference is the LP relaxation of the standard model that features the combined set of big-M Constraints (6.5)+(6.6). The results for this reference model are listed in the column titled "none". Adding valid inequalities (6.1) to the reference model affects only a minor improvement: The objective function value increases by 1.06 % on average. Valid inequality (6.3) lifts the lower bound to a larger extent; it causes an average increase of 6.74 %. The combination of inequalities (6.1) and (6.3), denoted by (6.1)+(6.3), does not expose substantial synergies in case of the LP relaxation.

The power of the valid inequalities is revealed in Tables 7.24 and 7.25, which display results for the MIP model with a time limit of 5 minutes and 1 hour, respectively. Especially valid inequalities (6.3) are effective. For the short time limit, inequalities (6.3) allow to solve *all* instances of the set $K = 20$, $P = 10$ to optimality within a very short average solution time of only 5.8 seconds (cf. Table 7.24).

Overall, the combination (6.1)+(6.3) has positive synergies in case of the MIP. For example, the combination allowed to solve one instance of the set $K = 50$, $P = 100$ to optimality (cf. Table 7.25). The combination featured relatively small gaps. An exception is the average gap for the test set $K = 50$, $P = 100$, where the best solution found after



(a) Instance 1 of the set $K = 20$, $P = 20$ from Table 7.18 (absolute gap: 1, relative gap: 2.99 %)



(b) Instance 1 of the set $K = 100$, $P = 50$ from Table 7.18 (absolute gap: 30, relative gap: 16.72 %)

Figure 7.4: Value of the best bound and objective function value of the incumbent solution during branch-and-cut for a small- and a medium-sized instance

1 hour is of poor quality in case of four instances. On the other hand, for one instance of this test set, the combination (6.1)+(6.3) provides the best solution across all models.

Table 7.23: Average optimal objective function values obj_μ and average solution times $time_\mu$ in brackets for the LP relaxation of the standard model with different globally valid inequalities for testbed 1 ($\rho^{\text{proj}} = 0.6$, 1 thread)

Instance set			Standard model, LP relaxation			
			obj_μ ($time_\mu$ [s])	for globally valid inequalities ...		
K	P	$D = S$	... none	(6.1)	(6.3)	(6.1)+(6.3)
10	10	3	15.3 (0.02)	15.8 (0.04)	17.4 (0.03)	17.5 (0.03)
10	20	3	24.5 (0.15)	24.6 (0.26)	26.4 (0.15)	26.4 (0.15)
20	10	3	25.7 (0.06)	26.2 (0.18)	29.1 (0.11)	29.1 (0.10)
20	20	5	40.2 (0.38)	40.4 (0.56)	43.1 (0.88)	43.2 (0.71)
50	100	10	267.0 (37.01)	267.0 (52.31)	269.0 (96.36)	269.0 (65.16)
100	50	10	167.7 (33.06)	168.4 (55.53)	170.8 (157.60)	171.4 (160.06)

In case of the larger-sized instances in Table 7.25, none of the four models dominates the other. For an example, consider the test set $K = 50$, $P = 100$. While the relative gap is smallest for (6.1)+(6.3) with 5.7 %, the average objective function value is best when none of the valid inequalities is applied. At the same time, each of the four models provides the best solution found for at least one of the ten instances out of this test set. Altogether, we recommend to apply the globally valid inequalities (6.1)+(6.3), although their benefit seems to vanish for larger-sized instances. From now on, these inequalities were applied, unless otherwise indicated.[14]

In Subsection 6.1.1, we also outlined locally valid inequalities, namely, inequalities (6.2) and (6.4). Each of these inequalities can be added as a local cut to the subproblem related to a node of the branch-and-cut tree if the solution to of the LP relaxation at that node violates the inequality. We implemented this cutting plane approach by two callback functions: one for (6.2) and one for (6.4). A callback function, also called cut callback, is invoked by the solver CPLEX during its branch-and-cut procedure. The callback function checks whether an inequality is violated. If so, the respective cut is added before the branch-and-cut procedure continues.

In case of the solver CPLEX, the use of cut callbacks requires special solver settings.

[14] When we ran the 64 bit version of CPLEX, which could exploit 32 GB RAM in our case, and replicated the experiments of Table 7.25, we obtained almost identical results, regardless of whether CPLEX 64 bit could use 1 thread or up to 8 threads. When we repeated the experiments of Table 7.24, CPLEX 64 bit yielded significant time savings for the first three test sets listed in this table, especially when using up to 8 threads.

Table 7.24: Numbers of proven optimal solutions *opt.*, average relative gaps gap_μ, and average solution times $time_\mu$ for the standard model with different globally valid inequalities for testbed 1 ($\rho^{\text{proj}} = 0.6$, 1 thread, search strategy: best bound)

Instance set			Standard model, $t^{\max} = 300\,\text{s}$			
			opt. $\langle gap_\mu\ [\%] \rangle$ ($time_\mu$ [s])	for globally valid inequalities ...		
K	P	$D = S$	... none	(6.1)	(6.3)	(6.1)+(6.3)
10	10	3	10	10	10	10
			⟨–⟩	⟨–⟩	⟨–⟩	⟨–⟩
			(1.7)	(1.4)	(0.2)	(0.3)
10	20	3	9	10	10	10
			⟨0.3⟩	⟨–⟩	⟨–⟩	⟨–⟩
			(60.3)	(64.2)	(49.5)	(60.1)
20	10	3	8	7	10	10
			⟨1.2⟩	⟨2.4⟩	⟨–⟩	⟨–⟩
			(89.0)	(95.8)	(5.8)	(8.7)
20	20	5	6	7	6	7
			⟨2.8⟩	⟨2.7⟩	⟨2.6⟩	⟨2.0⟩
			(164.9)	(155.2)	(161.8)	(153.8)

Table 7.25: Numbers of proven optimal instances *opt.*, average relative gaps gap_μ, and average solution times $time_\mu$ for the standard model with different globally valid inequalities for testbed 1 ($\rho^{\text{proj}} = 0.6$, 1 thread, search strategy: best bound)

Instance set			Standard model, $t^{\max} = 1\,\text{h}$			
			opt. $\langle gap_\mu\ [\%] \rangle$ ($time_\mu$ [s])	for globally valid inequalities ...		
K	P	$D = S$	... none	(6.1)	(6.3)	(6.1)+(6.3)
20	20	5	6	8	8	9
			⟨1.5⟩	⟨0.9⟩	⟨0.8⟩	⟨0.5⟩
			(1484.9)	(1016.3)	(881.5)	(603.2)
50	100	10	0	0	0	1
			⟨7.6⟩	⟨6.0⟩	⟨7.4⟩	⟨5.7⟩
			(3600.0)	(3600.0)	(3600.0)	(3569.2)
100	50	10	0	0	0	0
			⟨21.6⟩	⟨21.6⟩	⟨23.0⟩	⟨73.1⟩
			(3600.0)	(3600.0)	(3600.0)	(3600.0)

We had to change the setting for the search method from the default *dynamic search* to *traditional branch-and-cut*.[15] For a fair comparison, we compared the (potential) use of locally valid inequalities to the case where only globally valid inequalities are applied and the search method is set to traditional branch-and-cut (cf. Table 7.26). Our results reveal that the search method *traditional branch-and-cut* performs considerably worse than the method *dynamic search* for some test sets.

We conducted preliminary tests in order to find an adequate frequency for searching violated locally valid inequalities and to determine which cuts to add to the model. For inequality (6.2), we concluded that it is adequate to check for violated inequalities every 10th node. For inequality (6.4), we inferred that searching at every node is suitable. Furthermore, we decided based on the tests to neither add only a maximally violated cut nor the first cut encountered, but all violated inequalities.[16]

We examined the effect of locally valid inequalities (6.2) and (6.4) on solution time and quality for a time limit of 1 hour. Table 7.26 shows for both (6.2) and (6.4) that their effect is positive for some test sets, but negative for others. Inequality (6.4) tends to be more effective than inequality (6.2). For the set $K = 20$, $P = 20$, the average number of local cuts added per instance amounts to 134 in case of (6.2) and to 867 in case of (6.4). Let us put these numbers in relation to the number of tries to find a violated inequality: There were 0.03 cuts added per callback in case of (6.2) and 0.04 in case of (6.4). The use of inequalities (6.4) outperformed the dynamic search method on average only in case of test set $K = 20$, $P = 10$. Altogether, both locally valid inequalities are no reliable support for solving the workforce assignment problem. We verified this result also for the combined use of both inequalities.

Having analyzed the performance impact of globally and locally valid inequalities, we go on with testing the impact of parameter settings. First, we will test the impact of the parameter ρ^{proj}. Finally, we consider the impact of instance size examining especially instances of large size.

Parameter ρ^{proj} influences the ratio of total project requirements to the corresponding availability of the workforce. As in the case of the project selection problem (cf. Figure 7.2 on page 214), we found that this parameter has a notable impact on solution time in case of the workforce assignment problem. However, the relationship between ρ^{proj} and solution time is different from the relationship that we observed for the project selection problem.

To scrutinize the impact of ρ^{proj}, we ran CPLEX for the standard model on the instances of testbed 3. The instances of testbed 3 are partitioned in 17 test sets. Each test set was generated with a distinct value of ρ^{proj}, which was chosen from the set $\{0.05, 0.1, 0.15, \ldots, 0.85\}$. When we solved the 17 test sets, the time limit of the solver was set to 1 hour for each instance.

The results for the test sets of testbed 3 are presented in Figure 7.5, which illustrates the impact of the parameter ρ^{proj} on the average solution time $time_\mu$ and on the average number of assignments denoted by obj_μ. As expected, the number of assignments increases with increasing workload. The sharp increase in average computation time for

[15] The *search method* (dynamic search vs. traditional branch-and-cut) must not be confused with the *search strategy* (best bound vs. depth-first, for example)

[16] It should be more effective to add violated inequalities at nodes at the same depth of the branch-and-cut tree. Unfortunately, the solver CPLEX does not readily support such a strategy.

Table 7.26: Average solution times $time_\mu$ [s], numbers of proven optimal solutions $opt.$, and average relative gaps gap_μ [%] for the standard model with different locally valid inequalities for testbed 1 ($\rho^{\text{proj}} = 0.6$, 1 thread, search strategy: best bound)

			Standard model, $t^{\max} = 1\,\text{h}$											
			Results for locally valid inequalities ...											
Instance set			... none (dyn. search)			none (trad. b.-a.-c.)			(6.2) (trad. b.-a.-c.)			(6.4) (trad. b.-a.-c.)		
K	P	$D = S$	$time_\mu$	$opt.$	gap_μ	$time_\mu$	$opt.$	gap_μ	$time_\mu$	$opt.$	gap_μ	$time_\mu$	$opt.$	gap_μ
10	10	3	0.3	10	–	0.3	10	–	0.3	10	–	0.3	10	–
10	20	3	60.3	10	–	336.4	10	–	301.5	10	–	255.9	10	–
20	10	3	8.7	10	–	9.4	10	–	22.1	10	–	4.1	10	–
20	20	5	603.2	9	0.47	1266.7	8	1.14	1350.3	7	1.75	1319.8	7	1.42
50	100	10	3569.2	1	5.75	3600.0	0	5.16	3600.0	0	6.15	3600.0	0	7.11
100	50	10	3600.0	0	73.09	3600.0	0	107.53	3600.0	0	108.01	3600.0	0	106.60

test sets with $\rho^{\text{proj}} \geq 0.55$ is rather counter-intuitive, though.[17] Since the number of feasible solutions, i.e., the number of feasible assignments $\boldsymbol{x}$, decreases with ρ^{proj}, one could expect solution time to decrease as well. However, the average number of nodes processed increases with ρ^{proj}. For all instances of the test set $\rho^{\text{proj}} = 0.3$, only the root node was processed, because heuristics that are applied by CPLEX during root node processing found a solution whose optimality could be verified. For $\rho^{\text{proj}} = 0.5$, 818 nodes per instance were processed on average, while 65 588 nodes were processed in case of $\rho^{\text{proj}} = 0.85$.

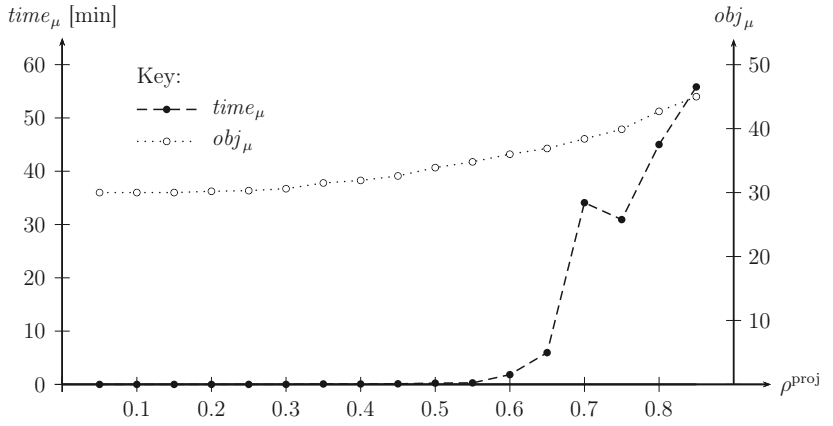


Figure 7.5: Average solution times $time_\mu$ and average numbers of assignments obj_μ for the test sets of testbed 3, which feature different ratios of project requirements to workforce availability ρ^{proj} ($K = 15$, $P = 15$, $D = S = 4$, $t^{\max} = 1\,\text{h}$, 1 thread, search strategy: best bound)

The difference in the number of nodes processed can be traced to at least two roots. First, the number of top-quality solutions increases with ρ^{proj}. Recall that a top-quality solution is a solution whose objective function value is not far away from the optimal objective function value. Second, the LP relaxation tends to become less tight when ρ^{proj} increases and additional assignments are necessary. We elaborate on these causes one by one.

To see how the number of top-quality solutions for the test sets $K = 15$, $P = 15$ increases with ρ^{proj}, we make a rough estimation of the maximum number of top-quality solutions for a small and a large value of ρ^{proj}. As can be seen from Figure 7.5, for a small value of ρ^{proj}, e.g., $\rho^{\text{proj}} = 0.1$, there are about two workers in a project team on average. Since each worker is suitable for every project, there are $\binom{15}{2} = 105$ possibilities to select 2 workers out of the workforce, which comprises 15 workers. Assuming that any pair (k, k') can accomplish the workload of all 15 projects, there are $105^{15} \approx 2.08 \cdot 10^{30}$ potential

[17] For the test sets $\rho^{\text{proj}} \geq 0.7$, the time limit was hit for some instances, i.e., for these instances a proven optimal solution could not be determined within 1 hour.

top-quality solutions. For a large value of ρ^{proj}, e.g., $\rho^{\text{proj}} = 0.8$, about three workers form a project team on average. Assuming that any triple (k, k', k'') can execute all 15 projects, there are $455^{15} \approx 7.42 \cdot 10^{39}$ potential top-quality solutions. Even if—due to limited availabilities and large project requirements—a considerable number of infeasible solutions is among these $7.42 \cdot 10^{39}$ solutions, the number of top-quality solutions is likely to increase with ρ^{proj}, until the average team size exceeds $\frac{K}{2}$ workers.

To see that the LP relaxation tends to become less tight when ρ^{proj} increases and additional assignments are necessary, regard the following example.

Example 7.3 Consider the case where $K = 3$, $P = 3$, $S = 3$, and $T = 1$. Given these data, we will construct two instances A and B. For our example, workers k_1 and k_2, projects p_1 and p_2, and skills s_1 and s_2 play the central role. Worker k_3, project p_3, and skill s_3 are only required to ensure that both instances A and B satisfy certain conditions. Instances A and B differ only in the requirements of p_1 and p_2, all other data are identical. Let $R_{k_1t} = R_{k_2t} = 10$ and let $l_{k_1s_1} = l_{k_2s_2} = 2$ and $l_{k_1s_2} = l_{k_2s_1} = 1$ for both instances. Let worker k_3 master each skill s with $l_{k_3s} = 2$ and let $R_{k_3t} = 20$. Project p_3 may require only skill s_3 with $r_{p_3s_3t} = 40$ so that the availability of k_3 is just sufficient to cover the workload of project p_3. Skill s_3 is required neither by p_1 nor by p_2. For instance A, $r_{p_1s_1t} = r_{p_1s_2t} = r_{p_2s_1t} = r_{p_2s_2t} = 1$ holds, and for instance B, all these requirements are equal to 8.

Next, we look at the lower bounds that are used in globally valid inequalities (6.1) and (6.3), especially at those lower bounds that are associated with p_1 and p_2. Due to worker k_3, these lower bounds are of identical value for instances A and B: $LB^{\text{glob}}_{p_1s_1} = LB^{\text{glob}}_{p_1s_2} = LB^{\text{glob}}_{p_2s_1} = LB^{\text{glob}}_{p_2s_2} = LB^{\text{glob}}_{p_1} = LB^{\text{glob}}_{p_2} = 1$.

From now on, let us consider only projects p_1 and p_2 and workers k_1 and k_2. Note that the total minimum number of assignments necessary for p_1 and p_2 is equal to 2 in case of instance A and equal to 4 in case of instance B. For instance A, the LP relaxation of the MIP model provides a tight lower bound of 2. For instance B, however, the relaxation is less tight: The optimal solution of the LP relaxation has an objective function value of 3. Table 7.27 shows for instances A and B a corresponding solution $(\boldsymbol{x}, \boldsymbol{y})$ of the LP relaxation.

For the solutions of both instances, which are given in Table 7.27, all the corresponding big-M constraints (6.6) can be satisfied with fractional values for variables x_{kp}. In case of instance A, starting from an optimal integer-feasible solution with $x_{k_1p} = 1$ and $x_{k_2p} = 0$, $p \in \{p_1, p_2\}$, a decrease in the value of x_{k_1p} must be compensated by an increase in x_{k_2p}, and vice versa. In case of instance B, however, starting from an optimal integer-feasible solution with $x_{k_1p} = 1$ and $x_{k_2p} = 1$, $p \in \{p_1, p_2\}$, this compensation with regard to variables x_{kp} is not required. Thus, a less tight LP relaxation results in case of instance B.□

To illustrate the relation between ρ^{proj} and the tightness of the LP relaxation, we recorded the optimal objective function values of the LP relaxation and of the corresponding MIP model for a set of 20 instances with different values of ρ^{proj}. To create the instances, we set $K = 3$, $P = 3$, $S = 3$, and $T = 1$. Each worker k has an availability of $R_{kt} = 30$ and masters all three skills. Skill levels $l_{k_1s_1}$, $l_{k_2s_2}$, and $l_{k_3s_3}$ are equal to 2, while all other skill levels are equal to 1. Each project p requires every skill. For instance a, $a \in \{1, 2, \ldots, 20\}$, we set $r_{pst} = a$, $p \in \mathcal{P}$, $s \in \mathcal{S}_p$, $t = 1$. This variation of the requirements r_{pst} is equivalent to varying ρ^{proj}; in fact, r_{pst} is proportional to ρ^{proj} here.

Table 7.27: Solutions for the LP relaxation of instances A and B in Example 7.3

Variable	Instance A	Instance B
$y_{k_1p_1s_1t}$	0.25	3
$y_{k_1p_1s_2t}$	0.5	2
$y_{k_1p_2s_1t}$	0.25	3
$y_{k_1p_2s_2t}$	0.5	2
$y_{k_2p_1s_1t}$	0.5	2
$y_{k_2p_1s_2t}$	0.25	3
$y_{k_2p_2s_1t}$	0.5	2
$y_{k_2p_2s_2t}$	0.25	3
$x_{k_1p_1}$	0.5	0.75
$x_{k_1p_2}$	0.5	0.75
$x_{k_2p_1}$	0.5	0.75
$x_{k_2p_2}$	0.5	0.75

Departmental requirements were neglected. To simulate weak lower bounds LB_{ps}^{glob} and LB_{p}^{glob}, we removed globally valid inequalities (6.1) and (6.3) from the LP relaxation. The results of the 20 instances, which are displayed in Figure 7.6, show that the LP relaxation is less tight whenever additional assignments of workers to projects are necessary.

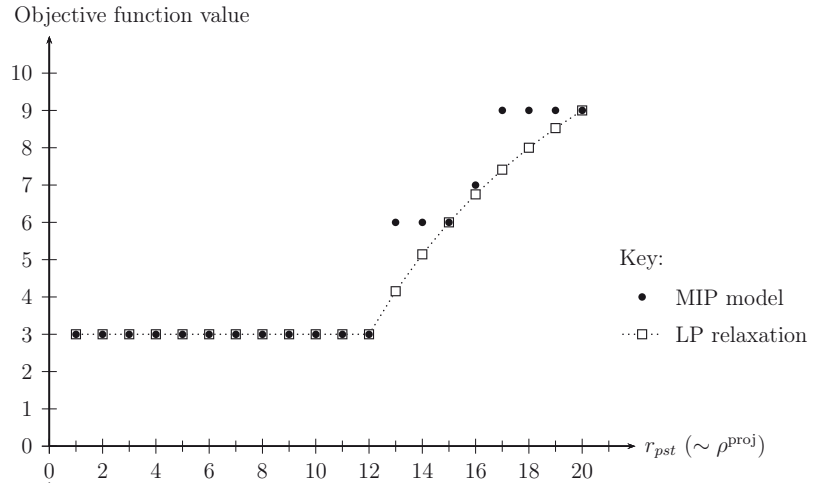


Figure 7.6: Optimal objective function values of a (weak) MIP model and the corresponding LP relaxation for 20 instances with different ratios of project requirements to workforce availability ρ^{proj} realized by different values for the project requirements r_{pst} ($K = 3$, $P = 3$, $D = 1$, $S = 3$, $T = 1$)

For the test sets $K = 15$, $P = 15$ from Figure 7.5, every increase in ρ^{proj} requires additional assignments. For each increase in ρ^{proj} from $\rho^{\text{proj}} = 0.5$ onwards, at least one additional assignment per instance is necessary on average. Especially for $\rho^{\text{proj}} \geq 0.7$, the deviation of the lower bound of the root node from the best integer-feasible solution found within 1 hour increased steadily. The weak LP-based lower bound is the second cause for the high number of nodes that must be processed in case of large values for ρ^{proj}.

In addition to the high number of nodes that must be processed, the time that is required to process a node increases with ρ^{proj}, because a larger amount of workload requires a greater number of variables y_{kpst} with positive values. Hence, the simplex algorithm tends to require more pivot operations. This increase in the number of pivot operations and the resulting increase in computation time can be observed for the test sets $K = 15$, $P = 15$ from Figure 7.5. On average, it takes 4 times more pivot operations and 18 times longer to solve the LP relaxation for $\rho^{\text{proj}} = 0.85$ than for $\rho^{\text{proj}} = 0.3$.

In summary, we noticed a serious drawback of the branch-and-cut approach. For increasing values of ρ^{proj}, we observed that computation time massively increases and, if a time limit is given, solution quality decreases. We call this phenomenon *the computational curse of high workforce utilization*. This phenomenon is of particular importance for firms and stresses the need for heuristic methods. Firms face real-life instances and these real-life instances are usually characterized by a high utilization of the workforce, because firms usually fully exploit capacities in order to realize the most profitable project portfolio. With regard to the parameter ρ^{proj}, utilization increases with ρ^{proj}. Hence, real-life instances tend to be just those instances that are relatively hard to solve. Solving these hard instances by a branch-and-cut solver is time-consuming, as our experiments confirmed. Thus, especially firms are in need of heuristic methods that provide solutions of equal or better quality in less time than a branch-and-cut solver.

Before we turn to the heuristics that we designed, we will present results obtained by CPLEX for the test sets of testbed 2, which contains instances of larger size. To be more precise, testbed 2 contains six test series. One test series contains instances of medium size, while five test series comprise instances of large size. The purpose of testbed 2 and the corresponding results is twofold. First, the results reveal up to which instance size CPLEX can provide solutions on a common personal computer. Second, the results serve as a basis for the assessment of our heuristic methods.

Recall that each of the six test series of testbed 2 comprises three test sets. The first test set of each series was generated with default parameter values and serves as a point of reference for the other two test sets of the series. The second test set of each series represents a more flexible workforce as, on average, the number of skills $|\mathcal{S}_k|$ that a worker k masters is greater for instances of this test set. The third test set was created with a greater value for ρ^{proj} resulting in a higher utilization of the workforce. Table 7.28 summarizes the results that we obtained for the test series of testbed 2 when we set the time limit of the solver CPLEX to 5 minutes and 1 hour.

For the test series $K = 40$, which comprises medium-sized instances, the improvement of objective function values that is achieved when the time limit is raised from 5 minutes to 1 hour is more pronounced for the second and third set than for the first set. Conversely, this means that solution quality for the second and third test set is rather low for the short time limit. In case of the second set where the workforce is more flexible, the large improvement points to the computational curse of flexibility: To obtain acceptable

Table 7.28: Average objective function values obj_μ and average relative gaps gap_μ of the standard model with different time limits $t^{\max}$ for the test sets of testbed 2 (1 thread, search strategy: best bound)

Instance set					Standard model			
					$t^{\max} = 300\,\text{s}$		$t^{\max} = 1\,\text{h}$	
K	P	$D = S$	$\lvert\mathcal{S}_k\rvert$	ρ^{proj}	obj_μ	gap_μ	obj_μ	gap_μ
40	40	6	1–3	0.6	112.6	21.4	105.7	11.2
			2–3	0.6	130.9	49.1	96.3	6.8
			1–3	0.8	234.2	125.0	127.5	18.5
200	100	20	1–3	0.6	*	*	1015.5	143.0
			3–5	0.6	16510.8	n.a.	4053.1	n.a.
			1–3	0.8	*	*	1217.3	151.5
400	150	30	1–3	0.6	*	*	1812.4	152.9
			3–5	0.6	*	*	*	*
			1–3	0.8	*	*	*	*
800	200	50	1–3	0.6	*	*	*	*
			3–5	0.6	*	*	*	*
			1–3	0.8	*	*	*	*
1250	300	60	1–3	0.6	*	*	*	*
			3–5	0.6	out of memory		out of memory	
			1–3	0.8	*	*	*	*
1500	500	75	1–3	0.6	out of memory		out of memory	
			3–5	0.6	out of memory		out of memory	
			1–3	0.8	out of memory		out of memory	

*For none of the ten instances, an integer-feasible solution was found within the time limit.

n.a. (not available): A gap value was not available for at least one instance.

solution quality, long computation times are necessary (cf. Table 7.10 on page 217 and the corresponding analysis as well as Walter and Zimmermann, 2012). In case of the third set, the computational curse of high workforce utilization is responsible for poor solution quality in the case where $t^{\max} = 5\,\text{min}$. This curse was also observed when the impact of the parameter ρ^{proj} was examined (cf. Fig. 7.5 on page 239).

For the first and third set of the test series $K = 200$, integer-feasible solutions were not found within the short time limit. In case of the second test set, only solutions of poor quality were found. The corresponding gap values could not be computed, because lower bound values were not available. The integer-feasible solutions were found by heuristics that CPLEX applies to the root node before the LP relaxation of this node is solved. For two instances, the poor heuristic solution could not be improved when the time limit was increased to 1 hour leading to an average objective function value of 4053.1. Without these two instances, the average objective function value amounts to 924.6 assignments. For the two instances with outlier objective values, a lower bound was not available even after

1 hour, because CPLEX spent the complete time on processing the root node inclusively applying heuristics.

For all instances with $K \geq 200$, root node processing was not finished until the time limit of 1 hour was reached. For these instances, the heuristics applied by CPLEX could provide integer-feasible solutions only for the second set of the test series $K = 200$. For all other test sets $K \geq 200$, an integer-feasible solution could not be determined within 1 hour.

Model size became critical for the test series $K = 1250$ and $K = 1500$. For the second set of the test series $K = 1250$ and for all sets of the series $K = 1500$, the memory required to load the model of each instance exceeded the available memory of 4 GB and caused CPLEX to report an out-of-memory error.[18]

To summarize the results of Table 7.28, we have to register that the exact solution method branch-and-cut, which we applied, is not suitable for solving large-sized instances and cannot even provide solutions of low quality in acceptable time for these instances. To tackle large-sized instances, heuristics might be a better choice. If the heuristics that we designed are a better choice is analyzed in the next subsection.

7.3.2 Analysis of heuristic methods

In this subsection, we analyze the performance of the heuristics that we developed for the workforce assignment problem. We present results for the heuristics GRAP and ModGRAP, ISAP, DROP, and ROUND, which were outlined in Subsections 6.2.1–6.2.4. For some of these heuristics, we compare variants of the respective heuristic, e.g., the variants of ISAP that result from different ways to calculate suitability values, and we compare alternative implementations, e.g., the two implementations of ISAP that differ in the way how the recurring problem of selecting workers is solved. Additionally, we consider for each heuristic the relation between the number of passes, i.e., the number of solutions constructed, and the quality of a best solution among those that were constructed. Furthermore, we consider the effect of parameters that determine important characteristics of an instance. The basis of our performance analysis are the three testbeds that we already used for testing the exact approach. As before, our criteria for evaluating performance are solution time and quality. Our goal of the analysis is to find out which heuristic among our four heuristics is the most suitable or, if an overall best heuristic does not exist, to find out when to apply which heuristic.

The following list summarizes which objects are analyzed in this subsection.

- Different heuristics: GRAP and ModGRAP, ISAP, DROP, ROUND
- For each heuristic:
  - Different variants (e.g., different suitability or unsuitability values)
  - Different implementations (e.g., usage of a general-purpose LP solver provided by CPLEX or of a specially tailored algorithm to solve subproblems)

[18] When we applied CPLEX 64 bit to the test series $K = 200$ and $K = 400$ with both $t^{\max} = 300\,\mathrm{s}$ and $t^{\max} = 1\,\mathrm{h}$, we obtained basically the same results as in Table 7.28, i.e., for many instances an integer-feasible solution could not be provided within the time limit. In contrast to CPLEX 32 bit, the 64 bit version did not suffer from memory shortage and could load the model for every instance of the test series $K = 1500$. Though, within 1 hour no solutions could be provided for these instances.

- The trade-off between the number of passes or solution time and solution quality
- Instance parameters: ρ^{proj}, instance size (K, D, P, S)

The first heuristics that we look at are GRAP and ModGRAP. GRAP staffs projects one at a time. For a project p, a roulette wheel process repeatedly selects an additional worker for p. The selection probability of a worker k in the roulette wheel selection is proportional to a suitability value $suit_{kp}$. Given the selected worker, as much of the remaining workload of project p as possible is allocated to him. Since this greedy randomized assignment procedure cannot guarantee to find any existing feasible solution, we developed a modified version of GRAP, which is called ModGRAP. ModGRAP *can* always find an existing feasible solution, though, its average performance is expected to be much worse than that of GRAP.

In a first experiment, we tested GRAP and ModGRAP against the test sets of testbed 1. For GRAP and ModGRAP, we set the number of passes, which is denoted by $pass^{\max}$, to 1000. Hence, the multi-start approach that was outlined in Algorithm 6.5 on page 126 tries 1000 times to construct a feasible solution and records a best solution encountered if a feasible solution can be constructed. In case of GRAP, we applied suitability value $suit^{\text{A}}_{kp}$, which is specified by Definitions (6.7), and also the second variant $suit^{\text{B}}_{kp}$, which is defined in Algorithm 6.8. Additionally, we tested a variant of GRAP where the probability of selecting a worker for a project p is equal for all workers $k \in \mathcal{K}_p^{\text{suit,rem}}$. This variant is denoted by GRAP($suit^{\text{rand}}_{kp}$), although we do not calculate a suitability value actually. The purpose of this variant with its random choice of a worker is to check whether the dedicated suitability values $suit^{\text{A}}_{kp}$ and $suit^{\text{B}}_{kp}$ have any positive impact. In case of ModGRAP, we report only on the outcomes for $suit^{\text{A}}_{kp}$. The results of the test runs for testbed 1 are summarized in Table 7.29. As a benchmark, the results obtained by the MIP solver of CPLEX for the standard model are also listed in this table.

From Table 7.29, we can see that GRAP provides solutions of low quality, but is very fast. With respect to solution quality, the utterly random choice of a worker for a project, which is marked by $suit^{\text{rand}}_{kp}$, is clearly outperformed by the variants that apply more sophisticated suitability values. GRAP is particularly fast when the procedure of selecting a worker for a project is rather simple, as it is the case for the variants that apply $suit^{\text{rand}}_{kp}$ or $suit^{\text{A}}_{kp}$. Suitability value $suit^{\text{B}}_{kp}$ is computationally more expensive than $suit^{\text{rand}}_{kp}$ and $suit^{\text{A}}_{kp}$, because the calculation of $suit^{\text{B}}_{kp}$ is more complex. The calculation requires sorting operations and processes more data. In exchange, using $suit^{\text{B}}_{kp}$ tends to provide better solutions. Though, the quality of these solutions exhibits a substantial gap to the solution quality obtained by CPLEX for the standard model. Averaged across all test sets of testbed 1, the deviation of objective function values of GRAP($suit^{\text{B}}_{kp}$) from the corresponding lower bounds that CPLEX provides for the standard model amounts to 51.0 %. The average deviation of CPLEX itself amounts to 13.2 %.

The solution quality of ModGRAP is much worse than that of GRAP. The deviation with respect to the best bounds provided by the exact approach averages to 304 %. ModGRAP could not outperform GRAP for a single instance. Computation times of ModGRAP($suit^{\text{A}}_{kp}$) are higher than that of GRAP($suit^{\text{A}}_{kp}$). This is mainly because the process of selecting a worker, which requires calculation of the suitability values, has

Table 7.29: Average solution times $time_\mu$ [s] and average objective function values obj_μ of GRAP and ModGRAP for testbed 1 compared to the results of the standard model, for which the average relative gaps gap_μ [%] are stated additionally

Instance set			$t^{\max} = 1\,\text{h}$			$pass^{\max} = 1000$							
			Standard model			GRAP($suit_{kp}^{\text{rand}}$)		GRAP($suit_{kp}^{\text{A}}$)		GRAP($suit_{kp}^{\text{B}}$)		ModGRAP($suit_{kp}^{\text{A}}$)	
K	P	$D = S$	$time_\mu$	obj_μ	gap_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ
10	10	3	0.3	18.5	–	0.02	25.2	0.03	23.2	0.09	22.7	0.07	52.6
10	20	3	60.3	28.5	–	0.04	42.2	0.06	39.7	0.17	38.3	0.14	101.7
20	10	3	8.7	30.3	–	0.04	41.9	0.07	39.1	0.25	37.7	0.14	89.9
20	20	5	603.2	47.1	0.47	0.08	78.0	0.12	69.2	0.36	69.3	0.28	174.1
50	100	10	3569.2	299.3	5.75	1.07	553.5	1.86	494.9	6.60	497.8	5.80	1520.3
100	50	10	3600.0	306.3	73.09	0.99	422.2	1.99	371.8	6.83	360.0	5.31	1014.9

to be passed more often in case of ModGRAP. More passes are necessary because per iteration ModGRAP allocates only the workload of *one* requirement r_{pst} to a worker.

In a second experiment, we tested GRAP against the larger-sized instances of testbed 2 (cf. Table 7.30). Again, we compared the alternatives $suit^{\mathrm{A}}_{kp}$ and $suit^{\mathrm{B}}_{kp}$ for the suitability values. In addition, we examined how solution quality changes when the number of passes is changed.

The results in Table 7.30 reveal two important properties of GRAP, one property being an advantage, the other being a drawback. The advantage is that GRAP can provide solutions for large instances within seconds or minutes, whereas a branch-and-cut approach based on the standard model cannot deliver an integer-feasible solution within an hour or cannot even load the model due to shortage of memory. On the other side, GRAP can fail to deliver a feasible solution when workforce utilization is relatively high, as can be seen from the entries for the six test sets with $\rho^{\mathrm{proj}} = 0.8$. Even if 1000 tries are undertaken to construct a solution, GRAP($suit^{\mathrm{A}}_{kp}$) fails for 5 out of 60 instances with $\rho^{\mathrm{proj}} = 0.8$ and GRAP($suit^{\mathrm{B}}_{kp}$) fails in case of 11 instances.

Another finding is that GRAP outperforms CPLEX for larger-sized instances given the time limit of 1 hour. For the first test set of the series $K = 200$ and $K = 400$, CPLEX found integer-feasible solutions within the time limit. These solutions of CPLEX are dominated by those of GRAP. For GRAP($suit^{\mathrm{B}}_{kp}$), the average objective function values are 14 % and 8 % lower than those of CPLEX for the first test set of the series $K = 200$ and $K = 400$, respectively. While the average gap in case of CPLEX amounts to 143 % and 153 %, respectively, the average gap of GRAP($suit^{\mathrm{B}}_{kp}$) with respect to the best bound determined by CPLEX amounts to only 109 % and 131 %, respectively.

In regard to workforce flexibility, we can observe that computation times of GRAP rise with increasing flexibility. When the average number of skills mastered by a worker increases, i.e., when the bounds of the range from which the number of skills $|\mathcal{S}_k|$ is selected for a worker k shifts towards greater values, more computational effort is necessary. Additional effort is necessary for selecting a worker k and for selecting a best matching skill during an iteration of GRAP in which workload is allocated to k. When a worker is selected for a project, more workers come into question when $|\mathcal{S}_k|$ increases. Hence, more suitability values must be calculated. When a best matching skill is selected, again, the number of candidates is the greater the greater $|\mathcal{S}_k|$. The additional computational effort is not compensated for by the reduced number of iterations per pass. The number of iterations reduces, because more flexible workers can cover a greater share of remaining project workload and thus enable a smaller number of assignments. Yet, GRAP cannot escape the curse of flexibility.

When the number of passes $pass^{\max}$ is reduced for GRAP, solution time decreases and solution quality declines. Solution time decreases proportionally. When reducing the number of passes from 1000 via 100 to 10, the number of assignments increases on average by 1.1 % and 2.7 %, respectively, compared to the number of assignments for 1000 passes. With respect to the 60 instances with $\rho^{\mathrm{proj}} = 0.8$, GRAP($suit^{\mathrm{B}}_{kp}$) failed to construct a feasible solution in case of 11 instances given 1000 passes. For 100 and 10 passes, GRAP($suit^{\mathrm{B}}_{kp}$) failed in case of 26 instances and 43 instances, respectively.

From the observation that GRAP does not reliably provide solutions for instances with a large value of ρ^{proj}, i.e., for instances with a high utilization of the workforce, we can draw two conclusions. First, GRAP is not an appropriate solution method when

Table 7.30: Average solution times $time_\mu$ [s] and average objective function values obj_μ of GRAP with different numbers of passes $pass^{\max}$ for testbed 2; outcomes are compared to the results of the standard model, for which the average objective function values obj_μ and the average relative gaps gap_μ [%] are stated

Instance set					Standard model		GRAP($suit^{\mathrm{A}}_{kp}$)		GRAP($suit^{\mathrm{B}}_{kp}$)					
					$t^{\max} = 1\,\mathrm{h}$		$pass^{\max} = 1000$		$pass^{\max} = 1000$		$pass^{\max} = 100$		$pass^{\max} = 10$	
K	P	$D = S$	$\lvert\mathcal{S}_k\rvert$	ρ^{proj}	obj_μ	gap_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ
40	40	6	1–3	0.6	105.7	11.2	0.6	172.6	1.7	168.3	0.2	171.9	0.02	178.7
			2–3	0.6	96.3	6.8	0.6	156.9	2.0	153.7	0.2	158.3	0.02	164.4
			1–3	0.8	127.5	18.5	0.6	222.8	1.8	211.5	**	**	**	**
200	100	20	1–3	0.6	1015.5	143.0	6.2	882.2	19.4	875.0	2.0	885.4	0.20	901.7
			3–5	0.6	4053.1	n.a.	11.7	756.2	34.2	744.7	3.5	752.8	0.35	766.7
			1–3	0.8	1217.3	151.5	**	**	**	**	**	**	**	**
400	150	30	1–3	0.6	1812.4	152.9	19.0	1673.3	58.8	1658.4	5.9	1676.7	0.60	1693.0
			3–5	0.6	*	*	40.9	1477.4	111.0	1468.2	11.1	1484.3	1.13	1510.5
			1–3	0.8	*	*	18.9	2181.7	**	**	**	**	**	**
800	200	50	1–3	0.6	*	*	68.7	3340.9	206.7	3312.1	20.7	3327.1	2.07	3365.2
			3–5	0.6	*	*	150.6	3002.1	388.4	2978.4	39.0	2995.5	3.96	3033.5
			1–3	0.8	*	*	**	**	**	**	**	**	**	**
1250	300	60	1–3	0.6	*	*	186.4	5389.9	502.4	5368.7	49.9	5398.6	5.13	5432.8
			3–5	0.6	out of memory		401.2	4888.6	938.8	4877.4	94.1	4909.8	9.38	4948.1
			1–3	0.8	*	*	**	**	**	**	**	**	**	**
1500	500	75	1–3	0.6	out of memory		441.7	8914.7	1018.6	9054.0	102.7	9088.9	10.28	9137.5
			3–5	0.6	out of memory		901.2	8075.4	1787.4	8189.2	178.7	8239.1	17.17	8296.3
			1–3	0.8	out of memory		509.3	10 910.8	1106.2	10 802.3	**	**	**	**

*For none of the ten instances, an integer-feasible solution was found within the time limit.

**For at least one out of the ten instances, a feasible solution could not be constructed within $pass^{\max}$ passes.

n.a. (not available): A gap value was not available for at least one instance.

workforce utilization is high. Especially firms should consider this drawback, because real-life instances faced by firms tend to feature a high utilization of the workforce. Second, manual planning with the goal of small project teams is an extremely difficult task given instances with a high level of workforce utilization. This second conclusion can be derived from the interpretation of GRAP as a proxy for a manual planning approach.

The results of GRAP for testbed 3, where ρ^{proj} is varied across test sets, are analyzed together with the results of ISAP for testbed 3. Before we turn to these results, we report on the performance of ISAP with respect to testbeds 1 and 2.

In each iteration of ISAP, all potential assignments (k, p), $p \in \mathcal{P}^{\text{toStaff}}$, $k \in \mathcal{K}_p^{\text{suit,rem}}$, are considered simultaneously. For each potential assignment (k, p), a suitability value $suit_{kp}$ is calculated. Then, for each project at most one worker is selected such that no worker is selected for more than one project, such that the number of selected pairs of workers and projects is as large as possible, and such that the total suitability of all selected pairs is maximized. For each selected pair (k, p), as much remaining workload of project p as possible is allocated to worker k.

The results for testbed 1 that we obtained when executing 1000 passes of ISAP are shown in Table 7.31. As suitability values we used $suit_{kp}^{\text{A}}$ and $suit_{kp}^{\text{B}}$. In order to test whether the suitability values $suit_{kp}^{\text{A}}$ and $suit_{kp}^{\text{B}}$ have a positive impact on solution quality at all, we also conducted runs with a random suitability value, which is denoted by $suit_{kp}^{\text{rand}}$. To determine this random suitability value for a pair (k, p), we drew a random number from a uniform distribution between 0 and 1 and assigned this random number to $suit_{kp}^{\text{rand}}$. For all three suitability values, we ran the variant ISAP(SSP). For the alternative variant ISAP(CPLEX), we present results only for $suit_{kp}^{\text{B}}$. The variant ISAP(CPLEX) calls the dual simplex method of CPLEX as LP solver to solve the subproblems of selecting workers for projects, whereas ISAP(SSP) applies the successive shortest path algorithm of Glover et al. (1986, pp. 12–19) to solve the recurrent selection problems. For both variants, the suitability values were perturbed after 100 passes according to Equation (6.18).

From Table 7.31, we made five observations. The first observation is that suitability values $suit_{kp}^{\text{A}}$ and $suit_{kp}^{\text{B}}$ clearly outperform the random suitability value $suit_{kp}^{\text{rand}}$ and that $suit_{kp}^{\text{B}}$ leads to substantially better solutions than $suit_{kp}^{\text{A}}$. For ISAP, there is a considerable difference in outcomes, whereas the advantage of $suit_{kp}^{\text{B}}$ over $suit_{kp}^{\text{A}}$ was much smaller in case of GRAP. A smaller number of assignments for $suit_{kp}^{\text{B}}$ than for $suit_{kp}^{\text{A}}$ goes hand in hand with a smaller number of selection problems that must be solved by ISAP($suit_{kp}^{\text{B}}$) than by ISAP($suit_{kp}^{\text{A}}$). Here, the reduction of calls of the solver for the selection problem is so significant that the advantage of $suit_{kp}^{\text{A}}$ with respect to computation time diminishes.

Second, solution quality of ISAP($suit_{kp}^{\text{B}}$) is better than that of GRAP($suit_{kp}^{\text{B}}$). However, for small-sized instances, solution quality of ISAP is still far worse than that of the exact approach. The time required per pass is higher for ISAP than for GRAP, since the solution of the recurrent selection problems places an additional burden on ISAP.

Third, although the same suitability value and the same seed for the random number generator were applied, ISAP(CPLEX, $suit_{kp}^{\text{B}}$) and ISAP(SSP, $suit_{kp}^{\text{B}}$) do neither provide identical solutions nor solutions of identical quality. This phenomenon can be attributed to different solutions of the selection problem. Within a pass, the solution process of both ISAP(CPLEX) and ISAP(SSP) is identical until the point where the solutions to a selection problem differ. The solutions do not differ in their objective function values, but in

Table 7.31: Average solution times $time_\mu$ [s] and average objective function values obj_μ of ISAP for testbed 1 compared to the results of the standard model, for which the average relative gaps gap_μ [%] are stated additionally

			$t^{\max} = 1\,\text{h}$			$pass^{\max} = 1000$							
			Standard model			ISAP(SSP)						ISAP(CPLEX)	
Instance set						$suit_{kp}^{\text{rand}}$		$suit_{kp}^{\text{A}}$		$suit_{kp}^{\text{B}}$		$suit_{kp}^{\text{B}}$	
K	P	$D = S$	$time_\mu$	obj_μ	gap_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ
10	10	3	0.3	18.5	–	0.1	24.7	0.1	22.2	0.2	21.0	1.1	21.0
10	20	3	60.3	28.5	–	0.5	42.8	0.5	38.8	0.6	35.2	2.2	34.8
20	10	3	8.7	30.3	–	0.4	41.6	0.4	35.7	0.6	32.3	2.3	32.4
20	20	5	603.2	47.1	0.47	0.5	77.1	0.5	64.4	0.9	59.2	4.6	59.2
50	100	10	3569.2	299.3	5.75	35.3	549.8	30.5	431.1	34.9	388.1	539.2	388.2
100	50	10	3600.0	306.3	73.09	15.2	416.5	15.4	305.2	18.9	254.8	398.9	254.7

their values of the variables. From such a point on, the solution processes of both variants proceed in two different directions that can lead to solutions of the workforce assignment problem that are of different quality. As one might expect, no variant dominates the other. For some instances, ISAP(CPLEX) found better solutions than ISAP(SSP); for other instances, ISAP(SSP) provided better solutions.

The fourth observation is the significant difference in computation times between ISAP(CPLEX) and ISAP(SSP). ISAP(SSP) is much faster than ISAP(CPLEX). The recurrent calls of the LP solver of CPLEX are on average nine times more time-consuming than the calls of the successive shortest path algorithm.

Finally, for the test set $K = 100$, ISAP($suit^{\mathrm{A}}_{kp}$) and ISAP($suit^{\mathrm{B}}_{kp}$) could outperform the exact approach, which was constrained by a time limit.

To facilitate a comparison between solution methods, we consider the average deviation of objective function values of ISAP(SSP, $suit^{\mathrm{B}}_{kp}$) from the corresponding lower bounds obtained for the standard model. The deviation amounts to 24.9 %. Compare this to the value of 51.0 % for the average deviation of GRAP($suit^{\mathrm{B}}_{kp}$) and to the deviation of 13.2 % for the exact approach.

Results of ISAP for testbed 2 are displayed in Table 7.32, where only the variant ISAP(SSP, $suit^{\mathrm{B}}_{kp}$) is considered, because $suit^{\mathrm{B}}_{kp}$ turned out to be the best choice as suitability value and solution times of ISAP(CPLEX, $suit^{\mathrm{B}}_{kp}$) are unacceptably high for large-sized instances. To keep solution times under 1 hour, we admitted only 100 passes. We examined the effect of reducing the number of passes from $pass^{\max} = 100$ to $pass^{\max} = 10$. Perturbations of suitability values according to Equations (6.18) were executed after 10 passes in case of $pass^{\max} = 100$ and after 5 passes in case of $pass^{\max} = 10$. For a convenient assessment, Table 7.32 lists also results of the exact approach for the standard model and of GRAP($suit^{\mathrm{B}}_{kp}$).

Table 7.32 shows that ISAP can provide better solutions than GRAP even if less passes for ISAP than for GRAP are executed. This implies that ISAP outperforms the exact approach for the test series $K \geq 200$ when a time limit of 1 hour must be observed. The average objective function values of ISAP(SSP, $suit^{\mathrm{B}}_{kp}$) are 41 % and 40 % lower than those of the MIP solver CPLEX in conjunction with the standard model for the first test set of the series $K = 200$ and $K = 400$, respectively. While the gap in case of CPLEX amounts to 143 % and 153 %, respectively, the gap of ISAP(SSP, $suit^{\mathrm{B}}_{kp}$) with respect to the best bound determined by CPLEX amounts to only 43 % and 52 %, respectively, whereas the gap of GRAP($suit^{\mathrm{B}}_{kp}$) amounts to 109 % and 131 %, respectively.

However, ISAP is also unreliable. Like GRAP, ISAP fails to construct a feasible solution for many instances where project workload is relatively high, i.e., where $\rho^{\mathrm{proj}} = 0.8$ holds.

In regard to workforce flexibility, we can record an interesting result: ISAP lifts the curse of flexibility. When the workers are more flexible, as they master more skills on average, computation time does not increase but decreases for ISAP. In case of ISAP, the number of assignments can be reduced significantly when flexibility increases. This significant reduction of assignments reduces the number of calls to the solver for the recurrent selection problem. The time saved is sufficient to compensate for the additional time that is necessary to compute more suitability values before each selection problem can be formulated. The additional time for computing suitability values is the main reason why GRAP suffers from the curse of flexibility, whereas ISAP can compensate

Table 7.32: Average solution times $time_\mu$ [s] and average objective function values obj_μ of ISAP with different numbers of passes $pass^{\max}$ for testbed 2; outcomes are compared to the results of the standard model and GRAP

Instance set					Standard model		ISAP(SSP, $suit^{\mathrm{B}}_{kp}$)				GRAP($suit^{\mathrm{B}}_{kp}$)	
					$t^{\max} = 1\,\mathrm{h}$		$pass^{\max} = 100$		$pass^{\max} = 10$		$pass^{\max} = 1000$	
K	P	$D = S$	$\lvert\mathcal{S}_k\rvert$	ρ^{proj}	obj_μ	gap_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ
40	40	6	1–3	0.6	105.7	11.2	0.4	130.7	< 0.1	131.1	1.7	168.3
			2–3	0.6	96.3	6.8	0.3	115.6	< 0.1	116.5	2.0	153.7
			1–3	0.8	127.5	18.5	**	**	**	**	1.8	211.5
200	100	20	1–3	0.6	1015.5	143.0	7.6	599.5	0.7	601.0	19.4	875.0
			3–5	0.6	4053.1	n.a.	7.4	438.4	0.7	441.5	34.2	744.7
			1–3	0.8	1217.3	151.5	**	**	**	**	**	**
400	150	30	1–3	0.6	1812.4	152.9	34.3	1090.2	3.2	1092.3	58.8	1658.4
			3–5	0.6	*	*	31.1	805.6	2.9	815.0	111.0	1468.2
			1–3	0.8	*	*	**	**	**	**	**	**
800	200	50	1–3	0.6	*	*	178.7	2126.9	16.8	2130.9	206.7	3312.1
			3–5	0.6	*	*	152.6	1593.5	14.6	1597.3	388.4	2978.4
			1–3	0.8	*	*	**	**	**	**	**	**
1250	300	60	1–3	0.6	*	*	662.7	3340.3	62.8	3344.7	502.4	5368.7
			3–5	0.6	out of memory		530.6	2484.6	50.7	2487.3	938.8	4877.4
			1–3	0.8	*	*	**	**	**	**	**	**
1500	500	75	1–3	0.6	out of memory		1909.3	5712.1	179.1	5726.9	1018.6	9054.0
			3–5	0.6	out of memory		1412.1	4180.6	132.5	4193.2	1787.4	8189.2
			1–3	0.8	out of memory		**	**	**	**	1106.2	10 802.3

*For none of the ten instances, an integer-feasible solution was found within the time limit.
**For at least one out of the ten instances, a feasible solution could not be constructed within $pass^{\max}$ passes.
n.a. (not available): A gap value was not available for at least one instance.

for this additional time and can avoid it partly, because less selection problems must be formulated and solved compared to instances with a less flexible workforce.

When the number of passes $pass^{\max}$ is decreased for ISAP, computation times fall proportionally and solution quality declines. When the number of passes is reduced from 100 to 10, the number of assignments increases slightly by 0.4 % on average. With respect to the 60 instances with $\rho^{\text{proj}} = 0.8$, ISAP(SSP, $suit^{\text{B}}_{kp}$) failed in case of 28 instances given 100 passes and in case of 40 instances given 10 passes.

As for GRAP, we can conclude for ISAP that this heuristic is adequate for firms only to a limited extent, because it can fail to provide feasible solutions, especially for instances with a high utilization of the workforce, which is typical for instances of firms.

This conclusion is confirmed by the results that we obtained for testbed 3, whose test sets differ in the amount of project workload and are thus characterized by different utilization rates for the workforce. The results of GRAP($suit^{\text{B}}_{kp}$) and ISAP(SSP, $suit^{\text{B}}_{kp}$) for testbed 3 are depicted in Figure 7.7.

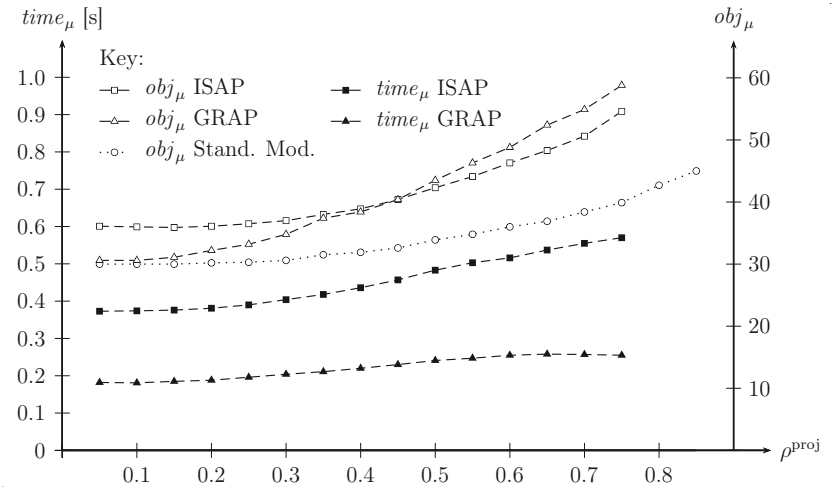


Figure 7.7: Average solution times $time_\mu$ and average objective function values obj_μ of GRAP($suit^{\text{B}}_{kp}$) and ISAP(SSP, $suit^{\text{B}}_{kp}$) with $pass^{\max} = 1000$ for testbed 3, average objective function values obj_μ of the standard model are plotted additionally ($K = 15$, $P = 15$, $D = S = 4$)

The data in Figure 7.7 approve our previous results. They indicate once more that GRAP and ISAP provide solutions that tend to be far from optimal and that GRAP and ISAP fail when utilization is high, although both methods undertake 1000 tries to construct a feasible solution for an instance. GRAP cannot construct a feasible solution for all instances of the test sets $\rho^{\text{proj}} = 0.8$ and $\rho^{\text{proj}} = 0.85$. ISAP cannot build a feasible solution for one instance of the test set $\rho^{\text{proj}} = 0.8$ and for four instances of the test set $\rho^{\text{proj}} = 0.85$.

Additionally, Figure 7.7 confirms that ISAP needs more time than GRAP, but provides

better solutions, in general. Only for small values of ρ^{proj}, GRAP constructs better solutions than ISAP. To see that the low solution quality provided by ISAP for small values of ρ^{proj} is a consequence of simultaneous assignments, consider the following example.

Example 7.4 Assume an instance of the workforce assignment problem with $K = 2$ and $P = 2$ where worker k_1 can completely cover both project p_1 and project p_2 and where worker k_2 can cover each of the two projects only partly. Let ISAP select worker k_1 for project p_1 and worker k_2 for project p_2. Worker k_1 must also be assigned to p_2 resulting in three assignments. GRAP, on the other hand, can construct a solution that features only two assignments, namely, those of worker k_1 to both projects. □

The solution times of GRAP and ISAP increase only slightly with ρ^{proj}. The increase is caused by additional iterations in which workers are assigned to projects. When ρ^{proj} increases, additional iterations are necessary, because more assignments are required. For example, ISAP had to solve on average 3.7 selection problems per instance for $\rho^{\text{proj}} = 0.2$ and 5.4 selection problems per instance for $\rho^{\text{proj}} = 0.6$. For the exact approach based on the standard model, we observed a massive increase in computation time for large values of ρ^{proj} (cf. Figure 7.5 on page 239). We termed this massive increase the computational curse of high workforce utilization. GRAP and ISAP do not suffer from this curse; their curse is unreliability.

Our drop method DROP, in contrast, is reliable. DROP starts with a feasible solution in which each worker $k \in \mathcal{K}$ is assigned to all her suitable projects $p \in \mathcal{P}_k^{\text{suit}}$. Then, assignments (k, p) are canceled until canceling any further assignment would result in an infeasible solution. To assess whether canceling an assignment (k, p) is feasible, we must check if all the associated remaining linear programs LP_t^{rem}, $t \in \mathcal{T}_p$, are feasible.

For the checks, we outlined two alternatives. One is to apply the dual simplex method of CPLEX as LP solver. Since the objective function of each remaining linear program is constant, CPLEX has to solve feasibility problems only. Alternatively, we can apply the generalized network simplex method for the checks. This method, however, requires a transformation of each remaining linear program into a generalized minimum cost flow problem whose objective function is not constant anymore. When the LP solver of CPLEX is used, the objective function of a remaining LP can be replaced by a surrogate objective function, which may help DROP to generate better solutions for the underlying workforce assignment problem. Objective function (6.22), which was introduced on page 187, is an example for a surrogate objective function.

When CPLEX is applied to solve the remaining linear programs, we can formulate such an LP either according to the standard model or according to the network model. We call the resulting solution methods DROP(CPLEX, stand.) and DROP(CPLEX, netw.), respectively. DROP(CPLEX) refers to both of these methods. The solution method that results when the generalized network simplex (GNS) is applied, is denoted by DROP(GNS) in the following.

For the implementation of the methods DROP(CPLEX), we considered two options to realize drop operations. In case of DROP(CPLEX, stand.), one option is to manipulate the right-hand side of Constraints (4.11), the other option is to exclude Constraints (4.11) from the remaining linear program and directly manipulate the upper bounds of the variables y_{kpst}. The latter option proved to be advantageous from a computational point of view, as solution times were considerably shorter for this option. Like-

wise, in case of DROP(CPLEX, netw.), we can either manipulate the right-hand side of Constraints (4.24) or exclude these constraints and directly manipulate the upper bounds of the variables f_{kpt}^{proj}. Again, we chose the second option.

Our assessment of the performance of DROP is divided in three parts. First, we compare the performance of DROP(CPLEX, stand.), DROP(CPLEX, netw.), and DROP(GNS) and look at the extensions of the simple drop scheme that we described in Subsection 6.2.3.1 and at an improvement of the generalized network simplex that we mentioned in Subsection 6.2.3.4. In the second part, we compare the alternative unsuitability values that we proposed for guiding the drop process. Additionally, we examine the effect of modifying the objective function of the remaining linear programs for DROP(CPLEX), i.e., we examine the effect of using the surrogate objective function (6.22). As explained in Subsection 6.2.3.5, objective function (6.22) cannot be used together with DROP(GNS). From the experiments in the first two parts, we deduce best variants of DROP and test them against the instances of testbeds 1 and 2, as we did before for GRAP and ISAP. Testbed 3 will be considered later together with ROUND.

In the first part of our performance analysis of DROP, we consider the variants DROP(CPLEX, stand.), DROP(CPLEX, netw.), and DROP(GNS) in conjunction with remaining linear programs whose objective function is not modified. In the first experiment of this part, we solved only the initial linear programs LP_t^{rem}, $t \in \mathcal{T}$. In DROP, these LPs are solved before the first drop operation is executed (cf. line 10 of Algorithm 6.11 on page 150). Our goal was to compare the efficiency with which the underlying LPs are solved. Here, we measure efficiency in terms of time. The solution times of the three variants are shown in Table 7.33 for the test sets of testbed 1 and for two test sets of testbed 2. The fastest variant is the variant that uses CPLEX to solve the initial remaining LPs based on the standard model. CPLEX in conjunction with the network model is the second fastest variant. Only for small-sized instances, it is outperformed by the variant that applies GNS. The time required by GNS increases drastically for larger-sized instances compared to CPLEX.

There may be several reasons for the inferiority of our generalized network simplex method GNS. The two reasons that are the most plausible for us shall be mentioned. One reason is that GNS requires substantially more pivot operations than CPLEX. For some instances of the set $K = 200$ in Table 7.33, GNS needs up to three times more pivot operations than CPLEX to solve an initial remaining linear program. Even if we use surrogate objective function (6.22) for the two variants of CPLEX, these variants require less pivot operations than GNS for the corresponding original objective function. Different pivoting rules may be a cause for the deviation in the number of pivot operations. Pivoting rules, also called pricing algorithms, can have a significant impact on the number of pivots and, hence, on solution time (cf. Chvátal, 1983, pp. 49–51; Pan, 2008). The default pivoting rule applied by CPLEX was steepest-edge pricing. In GNS, we used the priority rule outlined in Subsection 6.2.3.3 on page 166. Values for the parameters of this rule were taken from Kolisch and Heimerl (2012, p. 119) and fine-tuned. However, there may be better pivoting rules for our GNS.

A second reason for the inferiority of GNS is that the LP solver of CPLEX is supposed to be implemented much better than our GNS from a technical point of view. The solver package CPLEX is a well-established commercial product that has been refined by experts for more than two decades, whereas our implementation of the generalized network simplex

Table 7.33: Average solution times $time_\mu$ [s] and average objective function values obj_μ of the solver CPLEX in conjunction with the standard model and the network model and of the generalized network simplex (GNS) method for solving the initial linear programs for test sets of testbed 1 and 2 ($\rho^{\text{proj}} = 0.6$, $\mathcal{S}_k$: 1–3)

			Solving the initial linear programs LP_t^{rem}, $t \in \mathcal{T}$, only					
			CPLEX				GNS	
Instance set			Standard model		Network model			
K	P	$D = S$	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ
10	10	3	< 0.001	95.7	0.003	95.7	0.001	95.7
10	20	3	0.003	189.0	0.005	189.0	0.003	189.0
20	10	3	0.003	191.7	0.005	191.7	0.003	191.7
20	20	5	0.005	356.4	0.007	356.4	0.009	356.4
50	100	10	0.044	4042.0	0.054	4042.0	0.489	4042.0
100	50	10	0.038	4037.5	0.057	4037.5	0.445	4037.5
200	100	20	0.100	11 078.2	0.130	11 078.2	2.648	11 078.2
400	150	30	0.205	26 969.5	0.315	26 969.5	12.346	26 969.5

may not be as efficient as possible. In addition, be aware that the generalized minimum cost flow formulation which underlies GNS needs—like the network model—more variables and constraints than the standard model.

When we take a closer look at GNS and analyze the performance of its various parts, we realized that the tree manipulations, i.e., Algorithms 6.18–6.21, are executed very fast. The most time-consuming function among those functions that are directly related to the tree updates was the implementation of Algorithm 6.20, which consumed on average 0.5 % of the total time required by GNS to solve the initial remaining linear programs. In contrast, four other functions required 97 % of the total time. These functions were Algorithm 6.17 for computing the changes in arc flows depending on the entering arc (48 %), the function that determines the maximum absolute change δ for the flow on the entering arc (22 %), the function that updates the flows on the arcs according to the value of δ (19 %), and Algorithm 6.16 for computing node potentials (8 %). The large share in total time for these four functions seems reasonable, as these functions are burdened with multiplications and divisions of floating point numbers.

The inferiority of GNS is partly compensated when integrated into the simple drop scheme without extensions. This can be seen from the first of the three parts in Table 7.34. The first part of this table compares for the three variants of DROP a version that applies neither the extension provided by Algorithm 6.13 nor the extension provided by Algorithms 6.14 and 6.15. In this version of DROP, a pair (k, p) is only removed from the list $\mathcal{C}$ when a drop operation has been executed for this pair. Hence, if an identical seed is used for all three variants and if an unsuitability value is applied that does not consider the values of the variables y_{kpst}, the best solutions found by these variants are identical, because the drop sequences are identical. With respect to solution times, DROP(GNS) outperforms the other two variants for small-sized instances and DROP(CPLEX, netw.)

also for larger-sized instances. Here comes the advantage of DROP(GNS) to light. This advantage is based on the underlying generalized minimum cost flow problem, which allows warm starts because a feasible solution of a remaining LP is always feasible for the subsequent remaining LP that is associated with the next drop operation. Solutions stay feasible because a drop operation modifies only arc costs within the network. The advantage of a warm start is a fast reoptimization.

Table 7.34: Average solution times $time_\mu$ [s] and average objective function values obj_μ of the DROP variants DROP(CPLEX, stand.), DROP(CPLEX, netw.), and DROP(GNS) with and without extensions for testbed 1 (unsuitability value: us^A_{kp}, $pass^{\max} = 100$)

Instance set			DROP(CPLEX)				DROP(GNS)	
			Standard model		Network model			
K	P	$D = S$	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ
			DROP without Algorithms 6.13, 6.14, and 6.15					
10	10	3	0.5	19.6	0.7	19.6	0.3	19.6
10	20	3	1.2	33.0	1.7	33.0	0.8	33.0
20	10	3	1.2	33.7	1.7	33.7	0.8	33.7
20	20	5	2.6	55.8	4.2	55.8	2.2	55.8
50	100	10	117.0	387.0	257.2	387.0	138.7	387.0
100	50	10	86.1	276.2	206.5	276.2	173.1	276.2
			DROP with Algorithm 6.13 but without 6.14 and 6.15					
10	10	3	0.2	23.0	0.2	23.0	0.1	23.6
10	20	3	0.3	37.8	0.4	36.1	0.3	36.7
20	10	3	0.4	35.7	0.4	36.0	0.4	37.0
20	20	5	0.6	61.1	0.9	61.6	0.9	60.0
50	100	10	9.9	414.9	32.5	393.9	51.4	398.9
100	50	10	8.5	281.7	29.7	284.0	51.7	287.2
			DROP with Algorithms 6.13, 6.14, and 6.15					
10	10	3	0.2	22.8	0.2	23.0	0.1	22.2
10	20	3	0.3	37.8	0.3	35.7	0.3	37.1
20	10	3	0.3	36.0	0.4	36.3	0.4	36.7
20	20	5	0.5	60.7	0.8	61.4	0.9	59.5
50	100	10	9.1	411.3	27.2	396.9	47.9	398.6
100	50	10	8.4	283.0	28.6	282.3	52.7	287.7

Nevertheless, DROP(GNS) is outperformed by DROP(CPLEX, stand.) for the larger-sized instances. The inferiority of DROP(GNS) can be explained by an observation from the experiments and by a systematic disadvantage of DROP(GNS). In our experiments, we observed that the number of remaining linear programs that must be solved in the course of DROP is higher for DROP(GNS) than for DROP(CPLEX). Across all test sets

of testbed 1, DROP(CPLEX, netw.) solved 1.6 % and DROP(GNS) even 25.4 % more remaining linear programs than DROP(CPLEX, stand.). These numbers indicate that for a drop operation concerning a pair (k, p) in case of DROP(CPLEX, stand.) less reoptimizations are necessary. A reoptimization for a period $t \in \mathcal{T}_p$ is not necessary if worker k does not contribute to project p in period t in the solution that preceded the drop operation. Unlike DROP(CPLEX), DROP(GNS) seems to scatter the workload of a project p in each period $t \in \mathcal{T}_p$ across the set of suitable workers that have not been dropped.

The systematic disadvantage of DROP(GNS) is that the size of the underlying networks does not significantly shrink even if assignments are dropped. Admittedly, whenever a variable x_{kp} is finally fixed to 0, arcs that cannot enter the basis anymore are deleted from the sets of nonbasic arcs which are associated with the networks of the periods $t \in \mathcal{T}_p$. However, the set of nodes that is associated with each period $t \in \mathcal{T}$ does not change in the course of DROP(GNS). The solver CPLEX can shrink the size of the underlying models to a larger extent by deleting all variables y_{kpst} that are associated with a dropped pair (k, p). Decreasing model size allows CPLEX to solve the remaining linear programs $\mathrm{LP}_t^{\mathrm{rem}}$ faster and faster in the course of DROP(CPLEX). The two advantages of DROP(CPLEX) become even more apparent if Algorithm 6.13 is integrated into the drop procedure.

Algorithm 6.13 accelerates DROP considerably. After every successful (re)optimization, Algorithm 6.13 seeks for pairs (k, p) in the list $\mathcal{C}$ where worker k does not contribute to project p at all in the current solution. If such pairs are found, they are deleted from $\mathcal{C}$ without executing a drop operation. The use of Algorithm 6.13 within the heuristic DROP speeds up all three variants of this heuristic significantly, as can be seen from the second part of Table 7.34. For both variants of DROP(CPLEX), solution times decrease by 79 % averaged over the test sets of testbed 1. The decrease for DROP(GNS) amounts to only 59 % on average. Compared to DROP(CPLEX), the scattering of workload across workers that we observed for DROP(GNS) reduces the number of pairs (k, p) that can be removed from the list $\mathcal{C}$ by Algorithm 6.13.

Numbers verify that the difference in the decrease in solution times between DROP(CPLEX) and DROP(GNS) is caused in great part by a difference in the number of pairs removed from the list $\mathcal{C}$ by Algorithm 6.13. The number of removals from $\mathcal{C}$ at the very beginning of DROP is the most important one. The more pairs (k, p) can be removed from $\mathcal{C}$ early on, the less drop operations must be executed and the faster model size decreases, especially in case of DROP(CPLEX). Hence, we compare the number of removals due to Algorithm 6.13 that occur in the preparatory part of DROP, which is outlined in Algorithm 6.11. The average number of removals per instance, averaged over the test sets of testbed 1, is for DROP(CPLEX, netw.) 2 % and for DROP(GNS) 20 % less than for DROP(CPLEX, stand.). Note that the best solutions found by the three variants can differ for this version of DROP, because drop sequences can differ when different sets of pairs (k, p) are removed from $\mathcal{C}$.

The price of the speed-up due to Algorithm 6.13 is a potential decline in solution quality. In our experiment, the number of assignments increased by about 9 % for each of the three variants. The increase in the number of assignments is a consequence of Algorithm 6.13, since canceling a pair (k, p) in the preparatory part of DROP means that an optimal solution with $x_{kp} = 1$ cannot be reached anymore.

The last part of Table 7.34 shows the results that were obtained for the fully extended

version of DROP which also includes Algorithms 6.14 and 6.15. These algorithms compute the lower bounds LB_p^{Drop} and LB_{ps}^{Drop}, respectively. With the help of these bounds, additional pairs (k, p) might be removed from $\mathcal{C}$. Applying these lower bounds decreases computation times of DROP(CPLEX, stand.), DROP(CPLEX, netw.), and DROP(GNS) by 6 %, 8 %, and 3 %, respectively. Since computing the dynamic bounds LB_p^{Drop} and LB_{ps}^{Drop} is time-consuming, but may help to save time not until enough assignments have been canceled, Algorithms 6.14 and 6.15 are more effective when used together with Algorithm 6.13 than when used without Algorithm 6.13. Note that the best solutions found can differ from those found when Algorithms 6.14 and 6.15 are not used, because removing pairs from the list $\mathcal{C}$ can change the sequence of drop operations. In our experiment, the best solutions found differ sometimes, but there was no significant change in solution quality.

The time-saving strategy for DROP(GNS) to delete irrelevant nonbasic arcs was applied when we conducted the test runs for Table 7.34. Whenever a variable x_{kp} is finally fixed to 0, this strategy deletes those arcs from the set of nonbasic arcs of each period $t \in \mathcal{T}_p$ that are associated with worker k and project p, because these arcs do no longer come into question to enter the basis. To see that this strategy helps to reduce computational effort, Table 7.35 compares computation times for the case where the strategy is applied with computation times for the case where it is not applied, i.e., where irrelevant nonbasic arcs are not deleted. Deleting these arcs decreases computation times by 13 % on average. Though, as we saw, this time saving did not make DROP(GNS) faster than the variant DROP(CPLEX, stand.), which performed best.

Table 7.35: Average solution times $time_\mu$ [s] and average objective function values obj_μ of DROP(GNS) without Algorithms 6.13, 6.14, and 6.15 for the strategies to delete and to not delete arcs from the set of nonbasic arcs if they cannot enter the basis anymore, comparison for the test sets of testbed 1 (unsuitability value: us_{kp}^A, $pass^{\max} = 100$)

Instance set			Strategy of DROP(GNS) for irrelevant nonbasic arcs			
			Deletion		No deletion	
K	P	$D = S$	$time_\mu$	obj_μ	$time_\mu$	obj_μ
10	10	3	0.27	19.6	0.30	19.6
10	20	3	0.75	33.0	0.85	33.0
20	10	3	0.83	33.7	0.92	33.7
20	20	5	2.16	55.8	2.44	55.8
50	100	10	138.73	387.0	171.13	387.0
100	50	10	173.11	276.2	201.22	276.2

The second part of our analysis of DROP begins with an experiment in which we examined the effect of using alternative unsuitability values us_{kp}. An unsuitability value us_{kp} is calculated for each pair (k, p) in the list $\mathcal{C}$ before a pair is selected for the next drop operation. The value of us_{kp} determines the selection probability of the pair (k, p). In our experiment, we compared the performance of DROP(CPLEX, stand.) in conjunction with the three promising suitability values that were defined in Equations (6.19)–(6.21)

in Subsection 6.2.3.1. The first unsuitability value us_{kp}^{A} is a static value, which does not change in the course of DROP, whereas us_{kp}^{B} and us_{kp}^{C} are dynamic unsuitability values. The value us_{kp}^{B} considers the current number of assignments of worker k and the levels of the matching skills between worker k and project p, but is independent from the current contribution of worker k to project p, whereas the value us_{kp}^{C} takes the values of the variables y_{kpst}, $s \in \mathcal{S}_{kp}^{\mathrm{match}}$, $t \in \mathcal{T}_p$, into account and hence depends on the current solution for $\boldsymbol{y}$.

To test whether the three unsuitability values have a positive impact on solution quality at all, we also conducted test runs in which the pairs (k, p) were randomly selected. In this case, the selection probability was the same for each pair $(k, p) \in \mathcal{C}$. We denote this selection strategy by us_{kp}^{rand}, although we did not calculate an unsuitability value for this utterly random selection strategy. The results that we obtained for the different unsuitability values are summarized in Table 7.36.

Table 7.36: Average solution times $time_\mu$ [s] and average objective function values obj_μ of DROP(CPLEX, stand.) for different unsuitability values us_{kp} for testbed 1 ($pass^{\max} = 100$)

Instance set			DROP(CPLEX, stand.)							
			us_{kp}^{rand}		us_{kp}^{A}		us_{kp}^{B}		us_{kp}^{C}	
K	P	$D = S$	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ
10	10	3	0.2	22.7	0.2	22.8	0.2	22.0	0.2	22.8
10	20	3	0.3	38.1	0.3	37.8	0.3	37.2	0.3	38.7
20	10	3	0.3	37.3	0.3	36.0	0.4	35.5	0.3	36.0
20	20	5	0.5	62.5	0.5	60.7	0.5	61.5	0.5	62.5
50	100	10	8.7	421.1	9.1	411.3	8.7	404.8	9.4	423.4
100	50	10	8.0	289.8	8.4	283.0	8.2	278.0	8.2	286.5

The figures in Table 7.36 reveal that the impact of promising unsuitability values on solution quality is only modest. For each of the promising selection strategies associated with us_{kp}^{A}, us_{kp}^{B}, and us_{kp}^{C}, the random selection strategy denoted by us_{kp}^{rand} provides better solutions for at least one test set. Nevertheless, in total, the unsuitability values us_{kp}^{A} and us_{kp}^{B} outperform us_{kp}^{rand}. The value us_{kp}^{B} performs best on average.

The weak performance of us_{kp}^{C} is not surprising. The definition of this unsuitability value is based on values of the variables y_{kpst}. However, the variables y_{kpst} are not part of the objective function of the remaining linear programs. Thus, feasible but arbitrary values are assigned to these variables what makes us_{kp}^{C} to a random selection strategy like us_{kp}^{rand}. This similarity of us_{kp}^{C} and us_{kp}^{rand} is reflected in similar solution quality. Suitability value us_{kp}^{C} should perform better if the variables y_{kpst} are explicitly considered in the objective functions of the remaining linear programs.

The surrogate objective function (6.22) takes the variables y_{kpst} explicitly into account and can replace the original objective function of a remaining linear program in case of DROP(CPLEX). Objective function (6.22) should not be combined with DROP(GNS), as we have shown in Example 6.12 in Subsection 6.2.3.5. The goal of objective function (6.22) is to favor solutions of a remaining linear program $\mathrm{LP}_t^{\mathrm{rem}}$, $t \in \mathcal{T}_p$, where

$y_{kpst} = 0$, $s \in \mathcal{S}_{kp}^{\text{match}}$, tends to hold if the number of matching skills between worker k and project p is relatively small. We examined the effect of using surrogate objective function (6.22) for DROP(CPLEX, stand.) and DROP(CPLEX, netw.). Again, the variant DROP(CPLEX, stand.) performed best. For this variant, we present the results for the different unsuitability values in Table 7.37.

Table 7.37: Average solution times $time_\mu$ [s] and average objective function values obj_μ of DROP(CPLEX, stand.) with surrogate objective function (6.22) for different suitability values us_{kp} for testbed 1 ($pass^{\max} = 100$)

Instance set			DROP(CPLEX, stand.) with objective function (6.22)							
			us_{kp}^{rand}		us_{kp}^{A}		us_{kp}^{B}		us_{kp}^{C}	
K	P	$D = S$	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ
10	10	3	0.2	22.0	0.2	21.7	0.2	21.2	0.2	21.2
10	20	3	0.3	34.2	0.3	34.0	0.4	34.6	0.4	35.6
20	10	3	0.4	35.5	0.4	35.5	0.4	34.8	0.4	35.1
20	20	5	0.7	55.9	0.7	55.9	0.7	56.4	0.7	55.1
50	100	10	9.0	353.3	9.4	352.4	9.2	351.2	9.4	352.3
100	50	10	11.4	261.4	11.5	263.0	11.9	257.5	11.7	261.7

The results in Table 7.37 show that the usage of surrogate objective function (6.22) leads to substantially better solutions than the usage of the original objective function (cf. Table 7.36). The impact of the choice of the unsuitability value is modest again. On average, the random selection strategy us_{kp}^{rand} is outperformed only by a narrow margin. In total, the unsuitability value us_{kp}^{B} performs best. The surrogate objective function allows unsuitability value us_{kp}^{C} to better tap its potential. On average, the value us_{kp}^{C} performs second best and reaches solutions that are almost as good as solutions reached with us_{kp}^{B} and slightly better than solutions attained with us_{kp}^{A} and us_{kp}^{rand}. The difference in solution times between all four unsuitability values is negligible. DROP spends the greatest share in total computation time by far for solving the remaining linear programs. This domination conceals variations in computation times that arise from different unsuitability values.

The analysis of DROP finishes with the third part, which considers results for testbeds 1 and 2. From the first two parts of our analysis, we concluded that DROP(CPLEX, stand.) in conjunction with surrogate objective function (6.22) and unsuitability value us_{kp}^{B} is the best variant of DROP. Hence, we ran this variant against the test sets of testbeds 1 and 2. Additionally, we ran the variant DROP(GNS) with unsuitability value us_{kp}^{B}. An advantage of this variant is that it is less memory-intensive, as will become apparent from the results for testbed 2.

For testbed 1, we ran both variants of DROP for 1000 and 100 passes. The corresponding results are shown in Table 7.38. As we have already seen before, DROP(CPLEX, stand.) with (6.22) clearly outperforms DROP(GNS) with respect to solution quality and time. Even the solution quality of DROP(CPLEX, stand.) with (6.22) after only 100 passes is better than that of DROP(GNS) after 1000 passes. For both variants, computation time is approximately proportional to the number of passes. When the number

of passes is decreased from 1000 to 100 passes, average objective function values increase by about 3 % for both variants.

Comparing the results of DROP(CPLEX, stand.) with (6.22) and those of ISAP($suit^{\mathrm{B}}_{kp}$) for testbed 1 (cf. Table 7.31 on page 250), the drop method has a slight advantage with respect to solution quality, although it does not dominate ISAP($suit^{\mathrm{B}}_{kp}$). For 1000 passes, though, it takes DROP(CPLEX, stand.) with (6.22) six times the computation time that is required by ISAP(SSP, $suit^{\mathrm{B}}_{kp}$). However, the big advantage of all DROP variants is that they reliably provide feasible solutions, whereas ISAP and also GRAP can fail to find an existing feasible solution.

As ISAP, both variants of DROP outperform the exact approach that is based on the standard model for the test set $K = 100$. Note that we imposed a time limit on the exact approach. Averaged over all test sets of testbed 1, the deviation of objective function values of DROP(CPLEX, stand.) with (6.22) for $pass^{\max} = 1000$ from the corresponding lower bounds provided by the exact approach amounts to 19.7 %. For ISAP(SSP, $suit^{\mathrm{B}}_{kp}$), the deviation amounted to 24.9 %; for the exact approach it amounted to 13.2 %.

For testbed 2, the results are displayed in Table 7.39. We ran DROP(CPLEX, stand.) with surrogate objective function (6.22) for 100 and 10 passes. DROP(GNS) was run for 10 passes only, because computation times would have been unacceptably long otherwise. For both variants of DROP, unsuitability value us^{B}_{kp} was used and a time limit of 1 hour was applied. DROP stopped when the time limit or the maximum number of passes was hit, whatever happened first. For DROP(CPLEX, stand.) with (6.22), the case that the time limit was hit before 100 passes could be completed occurred only for two instances of the test set $K = 1250$, $\rho^{\mathrm{proj}} = 0.8$. In these two cases, 95 and 92 passes could be completed. For DROP(GNS), however, even 10 passes could not be completed within 1 hour for some larger-sized instances. In case of DROP(GNS), we report for each test set the average number of passes that had been completed before the time limit was reached. This average number of completed passes is denoted by $pass_{\mu}$.

Also for testbed 2, DROP(CPLEX, stand.) with (6.22) clearly outperforms DROP(GNS) in terms of solution time and quality. Though, for three out of four test sets for which DROP(CPLEX, stand.) with (6.22) fails due to excessive memory demand, DROP(GNS) does not fail. The memory demand of DROP(GNS) exceeds the available memory of 4 GB only in case of the second set of the test series $K = 1500$. This test set features a relatively flexible workforce and, hence, contains the largest-sized instances of testbed 2.

When the results of DROP are compared to the results of ISAP and GRAP, two observations can be made. First, DROP provides solutions for test sets with $\rho = 0.8$. The instances of these test sets are characterized by a high workforce utilization. ISAP and GRAP could not provide solutions for many of these instances.

The second observation concerns the remaining test sets which feature $\rho = 0.6$. With respect to solution quality, the results are ambivalent for these test sets. ISAP(SSP, $suit^{\mathrm{B}}_{kp}$) outperforms DROP(CPLEX, stand.) with (6.22) for those test sets where the instances feature a very flexible workforce, i.e., where $|\mathcal{S}_k| \in \{3, 4, 5\}$, $k \in \mathcal{K}$, while it is the other way round for those test sets with $|\mathcal{S}_k| \in \{1, 2, 3\}$, $k \in \mathcal{K}$. DROP(GNS), however, is clearly outperformed by ISAP(SSP, $suit^{\mathrm{B}}_{kp}$). On the other hand, DROP(GNS) outperforms GRAP($suit^{\mathrm{B}}_{kp}$) for $\rho = 0.6$ and $\rho = 0.8$. This is especially noteworthy for the test set $K = 1500$, $\rho = 0.8$, where DROP(GNS), GRAP, and a variant of ROUND were the only

Table 7.38: Average solution times $time_\mu$ [s] and average objective function values obj_μ of DROP for testbed 1 compared to the results of the standard model, for which the average relative gaps gap_μ [%] are stated additionally (unsuitability value: us^{B}_{kp})

Instance set			Standard model			DROP(CPLEX, stand.), (6.22)				DROP(GNS)			
			$t^{\max} = 1\,\mathrm{h}$			$pass^{\max} = 1000$		$pass^{\max} = 100$		$pass^{\max} = 1000$		$pass^{\max} = 100$	
K	P	$D = S$	$time_\mu$	obj_μ	gap_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ
10	10	3	0.3	18.5	–	1.6	20.4	0.2	21.2	1.2	21.5	0.1	22.1
10	20	3	60.3	28.5	–	3.3	33.1	0.4	34.6	3.1	35.2	0.3	36.5
20	10	3	8.7	30.3	–	3.8	34.3	0.4	34.8	3.6	35.0	0.3	36.2
20	20	5	603.2	47.1	0.47	6.6	54.1	0.7	56.4	8.6	57.1	0.9	59.9
50	100	10	3569.2	299.3	5.75	89.7	344.9	9.2	351.2	440.8	389.2	45.5	398.8
100	50	10	3600.0	306.3	73.09	112.7	253.5	11.9	257.5	477.9	279.6	48.5	285.4

Table 7.39: Average solution times $time_\mu$ [s], average objective function values obj_μ, and average numbers of completed passes $pass_\mu$ of DROP(CPLEX, stand.) with different numbers of passes $pass^{\max}$ and of DROP(GNS) for testbed 2; outcomes are compared to the results of the standard model (unsuitability value: us^{B}_{kp})

					$t^{\max} = 1\,\mathrm{h}$								
					Standard model		DROP(CPLEX, stand.), (6.22)				DROP(GNS)		
Instance set							$pass^{\max} = 100$		$pass^{\max} = 10$		$pass^{\max} = 10$		
K	P	$D = S$	$\lvert\mathcal{S}_k\rvert$	ρ^{proj}	obj_μ	gap_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	$pass_\mu$	obj_μ
40	40	6	1–3	0.6	105.7	11.2	3.3	137.2	0.4	140.6	0.9	10.0	140.6
			2–3	0.6	96.3	6.8	3.0	120.3	0.4	123.6	1.2	10.0	127.8
			1–3	0.8	127.5	18.5	4.4	161.2	0.5	166.7	1.1	10.0	163.2
200	100	20	1–3	0.6	1015.5	143.0	36.6	572.8	4.3	586.2	32.4	10.0	698.8
			3–5	0.6	4053.1	n.a.	52.4	444.6	6.2	453.0	79.9	10.0	556.8
			1–3	0.8	1217.3	151.5	62.8	717.0	6.9	739.0	36.3	10.0	821.2
400	150	30	1–3	0.6	1812.4	152.9	138.5	1040.3	17.1	1074.4	160.4	10.0	1325.7
			3–5	0.6	*	*	201.3	838.5	24.1	861.7	366.3	10.0	1105.8
			1–3	0.8	*	*	259.4	1340.6	28.9	1373.9	173.8	10.0	1576.3
800	200	50	1–3	0.6	*	*	588.7	2035.7	74.5	2085.8	779.6	10.0	2697.4
			3–5	0.6	*	*	857.7	1661.6	103.1	1700.1	2085.7	10.0	2224.2
			1–3	0.8	*	*	1132.9	2667.4	126.6	2725.1	863.2	10.0	3168.6
1250	300	60	1–3	0.6	*	*	1799.5	3209.1	226.3	3324.4	3253.6	9.8	4386.1
			3–5	0.6	out of memory		out of memory		out of memory		3600.0	2.6	3678.5
			1–3	0.8	*	*	†3377.0	†4183.5	389.7	4271.7	3476.7	9.3	5169.7
1500	500	75	1–3	0.6	out of memory		out of memory		out of memory		3600.0	1.0	7481.4
			3–5	0.6	out of memory		out of memory		out of memory		out of memory		
			1–3	0.8	out of memory		out of memory		out of memory		3600.0	0.7	9443.5

*For none of the ten instances, an integer-feasible solution was found within the time limit.
†For some instances out of the ten instances, 100 passes could not be completed within the time limit.
n.a. (not available): A gap value was not available for at least one instance.

methods that could provide feasible solutions. For this test set, DROP(GNS) provides the best solutions. The solutions of DROP(GNS) are better than those of GRAP($suit^{\mathrm{B}}_{kp}$) for nine out of ten instances, although DROP(GNS) could not complete a single pass in the case of three instances and although GRAP($suit^{\mathrm{B}}_{kp}$) had 1000 tries to construct a solution.

We also compare the heuristic DROP to the exact approach that applies the MIP solver of CPLEX to the standard model. The average objective function values of DROP(CPLEX, stand.) with (6.22) are 44 % and 43 % lower than those of the exact approach for the first test set of the test series $K = 200$ and $K = 400$, respectively. When the MIP solver of CPLEX tackles the standard model, the gap for these two test sets amounts to 143 % and 153 %, respectively. The gap of DROP(CPLEX, stand.) with (6.22) with respect to the best bound determined for the exact approach amounts to only 37 % and 45 %, respectively. For comparison, the gap of ISAP(SSP, $suit^{\mathrm{B}}_{kp}$) amounts to 43 % and 52 %, respectively.

When the number of passes is decreased from 100 to 10 for DROP(CPLEX, stand.) with (6.22), computation time decreases proportionally. Objective function values increase by approximately 3 % averaged over all test sets of testbed 2.

In regard to workforce flexibility, DROP suffers from the curse of flexibility. If workforce flexibility, which is associated with $|\mathcal{S}_k|$, $k \in \mathcal{K}$, is increased, computation time increases considerably, as Table 7.39 reveals. The more skills a worker masters on average, the more potential assignments exist, i.e., the more pairs (k, p) belong to the list $\mathcal{C}$ at the outset of DROP. However, after the initial remaining linear programs are solved, more pairs (k, p) can be removed from $\mathcal{C}$ due to zero contribution of worker k to project p. We observed that less drop operations are executed and less remaining linear programs are solved in case of greater flexibility. The reason for higher computation times is that more time is required to solve a remaining linear program due to increased model size. Greater flexibility implies that a worker masters more skills and can, hence, contribute to more skill requirements and perhaps to more projects. This in turn means that a remaining LP contains more variables y_{kpst} and may comprise more variables x_{kp} and more constraints.

When workforce utilization is increased, i.e., when instances with $\rho = 0.8$ instead of $\rho = 0.6$ are considered, solution time rises substantially in case of DROP(CPLEX, stand.) with (6.22). Two causes are responsible for this rise. First, more pairs (k, p) remain on the list $\mathcal{C}$ after the initial removal and, hence, more drop operations must be executed, i.e., more remaining LPs must be solved. Second, solving a remaining LP is more time-consuming than in case of a relatively low workforce utilization. We have already observed this effect for the LP relaxations in case of the exact approach (cf. Figure 7.5 on page 239 and the corresponding analysis). For the test series $K = 200$ in Table 7.39, for example, where the average computation time increases from 36.6 seconds to 62.8 seconds when utilization is raised, the average number of remaining LPs that must be solved per instance increases by 37 % and the time for solving the initial remaining LPs increases by 61 %. Thus, we must record that DROP(CPLEX, stand.) with (6.22) suffers from the computational curse of high workforce utilization.

In conclusion, DROP(CPLEX, stand.) with (6.22) is a solution method that reliably provides relatively good solutions in acceptable time as long as instance size is not too large such that memory limits are reached.

ROUND is another reliable heuristic that we developed for the workforce assignment

problem. In ROUND, the LP relaxation of a MIP formulation for the workforce assignment problem is repeatedly solved and manipulated until a solution of the LP relaxation is reached where all variables x_{kp}, $k \in \mathcal{K}$, $p \in \mathcal{P}_k^{\text{suit}}$, take on integer values. Given the solution of the LP relaxation, the manipulation works as follows: Each variable x_{kp}, $k \in \mathcal{K}$, $p \in \mathcal{P}_k^{\text{suit}}$, whose value is integer in the given solution is fixed to this value. Additionally, for each project p that features variables x_{kp}, $k \in \mathcal{K}_p^{\text{suit}}$, with fractional values in the given solution of the LP relaxation, the variable taking on the largest fractional value, say variable $x_{k'p}$, or the variable taking on the second largest fractional value, say $x_{k''p}$, is rounded up and fixed to 1. However, rounding up and fixing of one of these two variables is not carried out if for both workers k' and k'' another variable in the given solution has already been rounded up and fixed to 1. The manipulation that we described does not include rounding down a variable of positive value and fixing it to 0. This is why the heuristic ROUND guarantees to determine a feasible solution if any exists. This guarantee makes ROUND a reliable heuristic.

We implemented several variants of ROUND. Two basic variants that we implemented are ROUND(CPLEX, stand.) and ROUND(CPLEX, netw.). The former variant applies the LP relaxation of the standard model, whereas the latter uses the LP relaxation of the network model. For a limited number of passes, ROUND(CPLEX, stand.) slightly outperformed ROUND(CPLEX, netw.) with respect to solution time. The best solutions found by both variants are not identical, but average solution quality is—generally speaking—the same. For some instances, ROUND(CPLEX, stand.) determines slightly better solutions than ROUND(CPLEX, netw.). For other instances, the reverse is true. In the following, we only consider the basic variant ROUND(CPLEX, stand.), because it is the faster variant.

We implemented different variants of the basic variant ROUND(CPLEX, stand.). These variants differ in the big-M constraints and in the globally valid inequalities that are used for the LP relaxation. We show results for three variants that represent the broad range of possible variants. These variants are named ROUND(Tight LP), ROUND(Weak LP, (4.11)), and ROUND(Weak LP, (4.18)). The variant ROUND(Tight LP) applies the combined big-M constraint set (6.5)+(6.6) as well as the globally valid inequalities (6.1) and (6.3) for its LP relaxation. When we tackled the MIP, the corresponding model performed best with respect to solution time and quality. The variant ROUND(Tight LP) provides a relatively tight LP relaxation; hence, we hope that this variant provides good solutions for the workforce assignment problem, because an optimal solution of a tight LP relaxation may come close to an optimal solution of the corresponding MIP.

On the other hand, solving a tight LP relaxation tends to require substantially more time than solving a weak LP relaxation, as we have seen from Table 7.16 on page 227 and from Table 7.23 on page 235. To analyze the trade-off between solution quality and solution time, we also tested the variants ROUND(Weak LP, (4.11)) and ROUND(Weak LP, (4.18)). The variant ROUND(Weak LP, (4.11)) applies big-M constraint set (4.11) instead of the combined set (6.5)+(6.6), i.e., it applies weaker big-M constraints. Moreover, it applies neither (6.1) nor (6.3), i.e., ROUND(Weak LP, (4.11)) forgoes globally valid inequalities that tighten the model formulation. The variant ROUND(Weak LP, (4.18)) does also without globally valid inequalities (6.1) and (6.3). This variant applies Constraints (4.18) as big-M constraints. These big-M constraints are even weaker than big-M constraints (4.11).

For testbed 1, results of the variants ROUND(Tight LP) and ROUND(Weak LP, (4.18)) are shown in Table 7.40. We ran ROUND(Tight LP) for 100 and for 10 passes in conjunction with a time limit of 1 hour. The variant ROUND(Weak LP, (4.18)) was run for 10 passes. The dual simplex method of CPLEX was used to solve the LP relaxations.

Let us first look at the results of ROUND(Tight LP) for $pass^{\max} = 100$. The solution quality of this variant is better than that of any other heuristic that we tested. ROUND(Tight LP) with $pass^{\max} = 100$ outperforms DROP(CPLEX, stand.) with (6.22) with $pass^{\max} = 1000$. The average deviation of objective function values of ROUND(Tight LP) with $pass^{\max} = 100$ from the corresponding lower bounds provided by the exact approach amounts to 12.7 %. This deviation is smaller than the deviation of 19.7 % for DROP(CPLEX, stand.) with (6.22) with $pass^{\max} = 1000$ and smaller than the deviation of 13.2 % for the exact approach itself. However, solution times of ROUND(Tight LP) are far worse than those of DROP(CPLEX, stand.) with (6.22), especially for larger-sized instances. On average, ROUND(Tight LP) could complete only two thirds of the 100 passes for the instances of the test sets $K = 50$ and $K = 100$ within 1 hour, whereas DROP(CPLEX, stand.) with (6.22) could perform 1000 passes for the instances of these test sets within 2 minutes.

When we reduced the maximum number of passes to 10, ROUND(Tight LP) provided still good solutions. Solution time fell proportionally and the number of assignments increased by 3 %, averaged over all instances of testbed 1. The solutions obtained by ROUND(Tight LP) for $pass^{\max} = 10$ are better than those of DROP(CPLEX, stand.) with (6.22) with $pass^{\max} = 100$ and, on average, even better than those of DROP(CPLEX, stand.) with (6.22) with $pass^{\max} = 1000$. Though, even if only 10 or less passes of ROUND(Tight LP) are executed, solution times are relatively high.

Significantly shorter computation times are achieved by the variant ROUND(Weak LP, (4.18)). The speed-up is paid for by a deterioration of objective function values. Solution times of ROUND(Weak LP, (4.18)) with $pass^{\max} = 10$ are similar to those of DROP(CPLEX, stand.) with (6.22) with $pass^{\max} = 100$, but solution quality is much worse. Unlike ROUND(Tight LP), DROP, and ISAP, the variant ROUND(Weak LP, (4.18)) does not provide better solutions for the test set $K = 100$ than the exact approach does within 1 hour.

To shed light on the difference in solution times between ROUND(Tight LP) and ROUND(Weak LP, (4.18)) in case of $pass^{\max} = 10$, we scrutinized the solution process of these two variants. For the test sets $K = 20$ and $K = 100$, we recorded the number of LP relaxations that had to be solved until an integer-feasible solution for the underlying workforce assignment problem was reached. In case of ROUND(Tight LP), 6.8 LP relaxations per pass had to be solved for the instances of test set $K = 20$ and 10.7 iterations of rounding were required for instances of the set $K = 100$. In case of ROUND(Weak LP, (4.18)), 11.9 and 17.1 LP relaxations, respectively, had to be solved. Even though ROUND(Tight LP) requires less iterations to reach an integer-feasible solution, it requires more time than ROUND(Weak LP, (4.18)). This implies that it takes much more time to solve a single LP relaxation in case of ROUND(Tight LP) than in case of ROUND(Weak LP, (4.18)). One reason for the higher computational effort of ROUND(Tight LP) is the greater number of constraints that the corresponding LP relaxation exhibits.

To understand, why ROUND(Tight LP) requires a smaller number of iterations to yield an integer value for each variable x_{kp}, we looked at the subsequent solutions for

Table 7.40: Average solution times $time_\mu$ [s], average objective function values obj_μ, and average numbers of completed passes $pass_\mu$ of ROUND(CPLEX, stand.) with a tight and a weak LP relaxation of the standard model and with different numbers of passes $pass^{\max}$ for testbed 1 compared to the results of the standard model, for which the average relative gaps gap_μ [%] are stated additionally

			$t^{\max} = 1\,\text{h}$									
			Standard model			ROUND(CPLEX, stand.)						
						Tight LP					Weak LP, (4.18)	
Instance set						$pass^{\max} = 100$			$pass^{\max} = 10$		$pass^{\max} = 10$	
K	P	$D = S$	$time_\mu$	obj_μ	gap_μ	$time_\mu$	$pass_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ
10	10	3	0.3	18.5	–	1.1	100.0	19.5	0.2	20.7	0.1	23.6
10	20	3	60.3	28.5	–	11.7	100.0	31.8	1.3	32.6	0.1	39.2
20	10	3	8.7	30.3	–	3.0	100.0	32.7	0.4	34.0	0.2	34.8
20	20	5	603.2	47.1	0.47	33.4	100.0	52.7	4.1	54.4	0.4	66.6
50	100	10	3569.2	299.3	5.75	3553.8	66.6	317.2	625.8	319.6	9.8	499.2
100	50	10	3600.0	306.3	73.09	3487.4	67.0	226.6	694.4	229.4	9.6	326.5

the LP relaxation within a single pass. In the solution of the initial LP relaxation, there are more variables that are equal to 1 in case of ROUND(Tight LP) than in case of ROUND(Weak LP, (4.18)). The number of variables that equal 0 is virtually the same for both ROUND(Tight LP) and ROUND(Weak LP, (4.18)). Since more variables are fixed to 1 for ROUND(Tight LP) after the initial solution has been obtained, more variables are equal to 0 in the second solution. Consequently, less fractional variables occur in the second solution in case of ROUND(Tight LP) facilitating ROUND(Tight LP) to reach an integer-feasible solution in a smaller number of iterations than ROUND(Weak LP, (4.18)).

To give an example, we consider the test set $K = 100$. In the initial solution of ROUND(Tight LP), on average, 10 variables x_{kp} are equal to 1, 3489 variables x_{kp} are equal to 0, and 539 variables x_{kp} are fractional. For ROUND(Weak LP, (4.18)), no variable equals 1, 3477 variables equal 0, and 561 variables are fractional. For the following numbers that refer to the solution of the second LP relaxation, we take only variables into account that were not fixed after the first solution had been obtained. For ROUND(Tight LP), the solution of the second LP relaxation features on average 4 additional variables that equal 1, 149 additional variables that equal 0, and 343 fractional variables. For ROUND(Weak LP, (4.18)), no additional variable is equal to 1, 56 additional variables are equal to 0, and 469 variables are fractional in the second solution. For ROUND(Tight LP), almost three times more variables x_{kp} are equal to 0 than for ROUND(Weak LP, (4.18)).

For testbed 2, we ran all three variants of ROUND(CPLEX, stand.). The variants ROUND(Tight LP) and ROUND(WEAK, (4.11)) were run for 1 pass only, in order to keep computation times in an acceptable range. The variant ROUND(Weak LP, (4.18)), which is faster, was run for 10 passes. The results of these runs are summarized in Table 7.41.

Table 7.41 unveils that the variants ROUND(Tight LP) and ROUND(WEAK, (4.11)) are not suitable for large-sized instances. ROUND(Tight LP) cannot complete a single pass for the second and the third test set of the series $K = 200$ and for all test sets $K \geq 400$ within 1 hour. For almost all instances for which ROUND(Tight LP) could not determine a feasible solution, the LP solver of CPLEX could not find a feasible solution to the initial LP relaxation within 1 hour. Only for two instances of the second test set of the series $K = 200$ and for four instances of the first test set of the series $K = 400$, the process of rounding up and fixing variables and reoptimizing was started, but it could not be completed. The demand for memory exceeded the available memory of 4 GB RAM for the second test set of the series $K = 800$ and for all test sets $K \geq 1250$. ROUND(Weak LP, (4.11)) could provide feasible solutions for larger-sized instances than ROUND(Tight LP), but also failed for all test sets $K \geq 800$.

Up to the points where these two variants fail to provide feasible solutions within the time limit, their solution quality is competitive; though, their computation times are relatively high. ROUND(Tight LP) outperforms DROP(CPLEX, stand.) with (6.22) with $pass^{\max} = 100$, and ROUND(Weak LP, (4.11)) returns almost as good solutions as DROP(CPLEX, stand.) with (6.22) with $pass^{\max} = 10$; however, ROUND(Weak LP, (4.11)) requires more time than DROP(CPLEX, stand.) with (6.22) with $pass^{\max} = 100$.

In order to determine solutions for the largest-sized instances of testbed 2 by a rounding procedure within the time limit, we had to resort to the variant ROUND(Weak LP, (4.18)). The corresponding LP relaxation is solved so fast that 10 passes could be completed within

Table 7.41: Average solution times $time_\mu$ [s], average objective function values obj_μ, and average numbers of completed passes $pass_\mu$ of ROUND(CPLEX, stand.) with different LP relaxations of the standard model and with different numbers of passes $pass^{\max}$ for testbed 2; outcomes are compared to the results of the standard model

					$t^{\max} = 1$ h								
					Standard model		ROUND(CPLEX, stand.)						
							Tight LP		Weak LP, (4.11)		Weak LP, (4.18)		
Instance set							$pass^{\max} = 1$		$pass^{\max} = 1$		$pass^{\max} = 10$		
K	P	$D = S$	$\lvert\mathcal{S}_k\rvert$	ρ^{proj}	obj_μ	gap_μ	$time_\mu$	obj_μ	$time_\mu$	obj_μ	$time_\mu$	$pass_\mu$	obj_μ
40	40	6	1–3	0.6	105.7	11.2	47.7	121.4	2.6	141.9	16.6	10.0	160.6
			2–3	0.6	96.3	6.8	81.6	113.3	2.2	130.5	7.0	10.0	145.0
			1–3	0.8	127.5	18.5	59.1	141.1	5.7	161.3	24.9	10.0	181.1
200	100	20	1–3	0.6	1015.5	143.0	1093.1	534.9	94.5	619.5	33.8	10.0	794.0
			3–5	0.6	4053.1	n.a.	*	*	207.0	514.1	112.7	10.0	639.3
			1–3	0.8	1217.3	151.5	*	*	357.4	711.5	51.2	10.0	895.0
400	150	30	1–3	0.6	1812.4	152.9	*	*	739.3	1113.4	135.5	10.0	1473.5
			3–5	0.6	*	*	*	*	2021.1	938.1	504.7	10.0	1235.5
			1–3	0.8	*	*	*	*	**	**	207.3	10.0	1671.8
800	200	50	1–3	0.6	*	*	*	*	*	*	608.2	10.0	2941.3
			3–5	0.6	*	*	out of memory		*	*	2875.7	10.0	2486.8
			1–3	0.8	*	*	*	*	*	*	936.4	10.0	3341.2
1250	300	60	1–3	0.6	*	*	out of memory		*	*	2793.6	10.0	4819.6
			3–5	0.6	out of memory		out of memory		out of memory		3600.0	2.6	4144.6
			1–3	0.8	*	*	out of memory		*	*	3600.0	7.5	5450.2
1500	500	75	1–3	0.6	out of memory		out of memory		out of memory		3600.0	3.4	8703.2
			3–5	0.6	out of memory		out of memory		out of memory		out of memory		
			1–3	0.8	out of memory		out of memory		out of memory		3600.0	1.9	9479.7

*For none of the ten instances, an integer-feasible solution was found within the time limit.
**For at least one out of the ten instances, a feasible solution could not be constructed within the time limit.
n.a. (not available): A gap value was not available for at least one instance.

the time limit for all instances but the very large-sized ones. And only for one test set, namely, for the second test set of the series $K = 1500$, ROUND(Weak LP, (4.18)) failed to provide feasible solutions because of a memory shortage. However, solution quality of ROUND(Weak LP, (4.18)) is not competitive. DROP(CPLEX, stand.) with (6.22) and DROP(GNS) provide better solutions in less time.

From comparing ROUND(WEAK, (4.11)) and ROUND(Weak LP, (4.18)) to the exact approach, we can also conclude that these two rounding variants are inferior to DROP(CPLEX, stand.) with (6.22). For the first test set of the test series $K = 200$ and $K = 400$, average objective function values of ROUND(WEAK, (4.11)) are 39 % lower than those of the exact approach. Average objective function values of ROUND(WEAK, (4.18)) are only about 20 % lower than those of the exact approach, whereas average objective values of DROP(CPLEX, stand.) with (6.22) are about 43 % lower. When the exact approach runs for 1 hour, the average gap for these two test sets amounts to 143 % and 153 %, respectively. The gap of ROUND(WEAK, (4.11)) with respect to the best bound determined by the exact approach amounts to 48 % and 55 %, respectively. The gap of ROUND(Weak LP, (4.18)) is equal to 90 % and 106 %, respectively. For DROP(CPLEX, stand.) with (6.22) the gap amounts to only 37 % and 45 %, respectively.

Like DROP, ROUND suffers from the curse of flexibility. When workforce flexibility, which is associated with the numbers of skills $|\mathcal{S}_k|$ mastered by workers $k \in \mathcal{K}$, is raised, computation time of ROUND(Weak LP, (4.18)) tends to increase substantially. This increase occurs, although the number of LP relaxations that must be solved per pass decreases. The reason for the increase in overall solution time is the rise in time required to solve a single LP relaxation. In case of greater workforce flexibility, model size increases, because an average worker can contribute to more skills and projects than before. Hence, the number of variables x_{kp} and y_{kpst} and the number of constraints tends to increase for ROUND with increasing flexibility.

Also the degree of workforce utilization, which is represented by parameter ρ^{proj}, has an impact on solution time. Solution time of ROUND(Weak LP, (4.18)) increases, when ρ^{proj} is increased from 0.6 to 0.8, but the increase in computation time is less sharp than the increase that is induced by higher workforce flexibility. An increase in ρ^{proj} does not change model size. Reasons for the rise in computation time are an increase in the time required to solve a single LP relaxation and a slight increase in the number of iterations per pass that are necessary to yield an integer-feasible solution.

To sum up, ROUND(Tight LP) provides very good solutions but is so time-consuming that it is not practical for larger-sized instances. The variant ROUND(Weak LP, (4.18)) is less time-consuming but still not very fast. Moreover, this variant is not competitive with respect to solution quality.

Eventually, we will analyze the results of DROP and ROUND for testbed 3 in order to examine the impact of workforce utilization on solution time and quality. We have already realized for DROP(CPLEX, stand.) with (6.22) and for ROUND(Weak LP, (4.18)) that computation times increase when ρ^{proj} is increased from 0.6 to 0.8. The 17 test sets of testbed 3 can provide more information on the impact of ρ^{proj}, because they were created for 17 different values of the parameter ρ^{proj}, which determines the ratio of project workload to total availability. The corresponding instances of the 17 different test sets

are identical except for the values of the project requirements r_{pst}. These requirements are the larger, the larger the value of ρ^{proj} is.

For testbed 3, we compare the best variants of DROP and ROUND to the exact approach. We executed 1000 passes of DROP(CPLEX, stand.) with (6.22) using unsuitability value us^{B}_{kp} and 100 passes of ROUND(Tight LP). The results of our experiments are depicted in Figure 7.8.

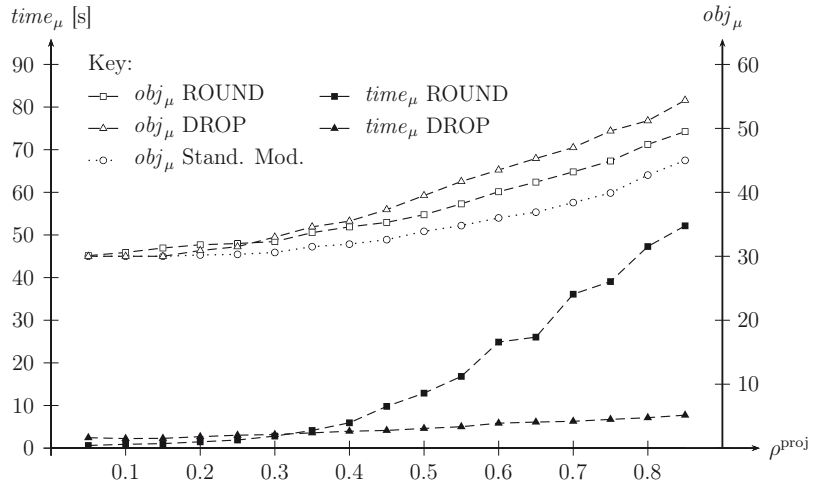


Figure 7.8: Average solution times $time_\mu$ and average objective function values obj_μ of DROP(CPLEX, stand., us^{B}_{kp}, $pass^{\max} = 1000$) with (6.22) and ROUND(Tight LP, $pass^{\max} = 100$) for testbed 3, average objective function values obj_μ of the standard model are plotted additionally ($K = 15$, $P = 15$, $D = S = 4$)

Figure 7.8 reveals that the solution quality of DROP(CPLEX, stand.) with (6.22) and ROUND(Tight LP) is better than that of GRAP and ISAP (cf. Figure 7.7 on page 253). ROUND(Tight LP) provides better solutions than DROP(CPLEX, stand.) with (6.22) except for the test sets $\rho^{\text{proj}} \leq 0.25$. We have noticed this advantage of ROUND(Tight LP) with respect to solution quality already for testbed 1.

The solution times of ROUND(Tight LP) and DROP(CPLEX, stand.) with (6.22) behave quite differently. For both heuristics, solution times increase with increasing ρ^{proj}. However, the increase in solution time for DROP(CPLEX, stand.) with (6.22) is moderate compared to the sharp rise in computation time for ROUND(Tight LP). This increase in computation time of ROUND(Tight LP) is also sharper than the increase that we observed for ROUND(Weak LP, (4.18)), when ρ^{proj} was raised from 0.6 to 0.8 in testbed 2.

We examined the discrepancy in solution times between DROP(CPLEX, stand.) with (6.22) and ROUND(Tight LP) for the test sets of testbed 3 in more detail by comparing the results for the test sets $\rho^{\text{proj}} = 0.3$ and $\rho^{\text{proj}} = 0.8$. For DROP(CPLEX, stand.) with (6.22), the average computation time per instance rises from 3.2 seconds to 7.1 sec-

onds. This increase is caused by a doubling in the number of drop operations, which comes along with a doubling in the number of remaining LPs that must be solved. While on average 41 remaining LPs must be solved per pass for $\rho^{\text{proj}} = 0.3$, 79 must be solved for $\rho^{\text{proj}} = 0.8$.

For ROUND(Tight LP), it takes 2.8 seconds on average to execute 100 passes for an instance of the test set $\rho^{\text{proj}} = 0.3$, but it takes 47.3 seconds in case of test set $\rho^{\text{proj}} = 0.8$. The average number of LP relaxations that must be solved per instance and per pass is nearly the same for both values of ρ^{proj}. It rises only slightly from 6.0 to 7.3. The cause for the drastic rise in computation time with increasing ρ^{proj} is the drastic increase in time that is required to solve a single LP relaxation.

The implications of these results are important when it comes to tackling real-life instances. Recall that instances with a high workforce utilization prevail in practice. Hence, the variant ROUND(Tight LP) may not be applicable in practice if time is a scarce resource. This and other implications of our results that we obtained for the different solution methods for the workforce assignment problem will be considered in the discussion in Chapter 8. Before starting the discussion, we will complete the numerical analysis with computational results for the utilization leveling problem, which emerges from a feasible solution of the workforce assignment problem.

7.4 Analysis for the utilization leveling problem

In this section, we consider computational results for the utilization leveling problem. First, we describe how we generated test instances for this problem. For the generated test instances, we compare the results of two solution methods. One method applies the dual simplex optimizer of CPLEX to determine an optimal solution for model (4.39)–(4.44). The simplex method, which is a general-purpose LP solver, has an exponential worst-case time complexity. The other method is Algorithm 6.4, which is a specially tailored leveling algorithm that runs in polynomial time.

We did not systematically create artificial instances for the utilization leveling problem, but simply considered feasible solutions for instances of the workforce assignment problem as instances of the utilization leveling problem. The resulting instances are artificial, too. The instances that we use for the performance analysis in this section were derived from solutions of DROP(CPLEX, stand.) with (6.22) for instances of testbed 2. To be more precise, the instances were derived from those solutions that were obtained for unsuitability value us^{B}_{kp} and $pass^{\max} = 100$ (cf. Table 7.39). The derived test sets are listed in Table 7.42. Since DROP(CPLEX, stand.) with (6.22) could not determine solutions for the second test set of the series $K = 1250$ and for all three test sets of the series $K = 1500$, the corresponding test sets are missing in Table 7.42. Each test set that is listed comprises ten files and each file comprises $D \cdot T$ instances of the utilization leveling problem. The ten files of a test set correspond to the ten instances of the associated test set for the workforce assignment problem.

In Table 7.42, we present results for both solution methods, which are denoted by CPLEX and Algorithm 6.4, respectively. For both methods, we report on objective function values and computation times that were averaged over the ten files of a test set. The value $time_{\mu}$ states the average time in milliseconds that was required to solve the $D \cdot T$

Table 7.42: Average objective function values obj_μ and average solution times $time_\mu$ [ms] of CPLEX for model (4.39)–(4.44) and of Algorithm 6.4 for instances of the utilization leveling problem that were derived from solutions for the corresponding instances of testbed 2 ($T = 12$)

Instance set					obj_μ	$time_\mu$ [ms]	
K	P	$D = S$	$\lvert\mathcal{S}_k\rvert$	ρ^{proj}		CPLEX	Algorithm 6.4
40	40	6	1–3	0.6	63 036.2	31	< 1
			2–3	0.6	58 407.5	31	< 1
			1–3	0.8	58 390.1	32	< 1
200	100	20	1–3	0.6	437 344.6	154	< 1
			3–5	0.6	435 077.6	153	< 1
			1–3	0.8	441 502.5	143	< 1
400	150	30	1–3	0.6	1 184 331.3	352	< 1
			3–5	0.6	1 183 607.4	340	< 1
			1–3	0.8	1 225 500.5	335	< 1
800	200	50	1–3	0.6	2 897 401.4	800	1
			3–5	0.6	2 925 118.5	775	1
			1–3	0.8	3 031 812.2	750	1
1250	300	60	1–3	0.6	6 059 730.3	1695	1
			1–3	0.8	6 387 157.1	1544	1

instances of a file from the test set under consideration. For each of the $D \cdot T$ instances of a file, we applied objective function (4.39) to calculate the objective function value of the optimal solution provided by the method under consideration. The value obj_μ represents the sum of these $D \cdot T$ objective function values, averaged over the ten files of the respective test set.

Both methods, i.e., the simplex method provided by CPLEX and Algorithm 6.4, determine solutions of identical quality. That is why the average cumulated objective function value obj_μ is stated only once. With respect to solution time, Algorithm 6.4 clearly outperforms CPLEX.

Chapter 8

Discussion

In this chapter, we discuss the results of the numerical analysis focusing on the hard problems of project selection and workforce assignment. Main results for the project selection problem were that an exact standard solver can provide good solutions even for large-sized instances, that marginal returns of multi-skilling are decreasing, and that skill chaining is advantageous, especially if diverse chains are implemented. Key results with respect to the workforce assignment problem were that only small-sized instances are computationally tractable by exact methods, that our tighter model formulation could accelerate the solution process but left medium- and large-sized instances still intractable, and that the heuristic DROP showed the overall best performance among all heuristics tested. Minor results refer to the impact of workforce utilization and multi-skilling on computation times. Throughout the following discussion, we will consider implications of our results for researchers and practitioners and we will point to limitations of our work. Eventually, we will indicate a further area where our approach can be applied.

The performance analysis for the project selection problem in Section 7.2 revealed a good computational behavior of the corresponding MIP model. Even when instance size was large, gaps of solutions were very small. Hence, we saw no need to outline heuristics. However, if instances of very large size are tackled, solution times may grow beyond acceptable limits. In this case, the high resolution of capacity supply, which results from modeling each single worker individually, becomes a burden, especially when the ratio ρ^{proj}, the ratio of capacity demanded from all selectable projects to the capacity supplied by the workforce, is in a medium range. Ratios ρ^{proj} that lie in this range are associated with longer computation times than low or high ratios ρ^{proj} (cf. Fig. 7.2 on page 214). For these burdensome instances, heuristics may be necessary. A starting point for a heuristic may be a MIP model that applies the aggregation of capacity supply outlined by Grunow et al. (2004, Section 3.3). Although this approach does not allow for heterogeneous skill levels and requires a constraint set whose size can be exponential in the number of skills, it may reduce computation times significantly and may lead to a good portfolio, which can possibly be improved by other methods.

Our formulation of the project selection problem has in particular two limitations. First, we consider only human resources. Other resources like budget, machinery, or other equipment are not considered. However, it would be simple to take these resources into account because the MIP model of the problem can be easily extended. Second, project selection is often a multi-objective problem, whereas our formulation features only a single objective and the solution approach is designed for the single-objective case. Though, solution methods that determine efficient solutions for a multi-objective problem are abundant and could be adopted to a multi-objective version of our problem.

M. Walter, *Multi-Project Management with a Multi-Skilled Workforce,* Produktion und Logistik, DOI 10.1007/978-3-658-08036-5_8, © Springer Fachmedien Wiesbaden 2015

In the test runs where we varied workers' flexibility by changing the number of skills per worker, we observed that marginal returns of flexibility decrease. The increase in portfolio benefit made possible by more flexible workers decreased with increasing flexibility of workers. This result is in line with the literature. Campbell (1999), Gomar et al. (2002), Vairaktarakis (2003), and Heimerl and Kolisch (2010a), for example, also observed diminishing returns of flexibility for their respective problems. Diminishing returns of flexibility have far-reaching implications for staff development strategies of human resource managers. A limited number of skills mastered by each worker is sufficient to cope well with fluctuations in skill requirements. The marginal additional benefits of fully flexible workers, who tend to be very expensive, render their employment not cost-efficient under most circumstances. Human resource managers should capitalize on these insights.

A further important implication of our work for human resource managers arises from the results concerning alternative skill configurations. When the number of skills per worker is limited but greater than one, it is crucial how skills are distributed across workers. Our experiments revealed that parallel chains derived from the classical chaining strategy of Jordan and Graves (1995) are a good but suboptimal choice. A strategy promoting diverse chains is a better choice to accommodate fluctuations in skill requirements. Our finding that diverse chains are superior to classical parallel chains confirms recent findings of Chou et al. (2011) who demonstrated that there are skill configurations which are superior compared to configurations based on classical chaining principles. For the first time, we determined superior configurations for a setting with a large number of parallel resources. Such a setting is typical for many firms. Implementing a skill configuration with diverse chains, however, may be more demanding than implementing a configuration with long parallel chains. It may be much easier to equip all members of a department with the same skills and design many identical chains across departments than to assign different skill sets to workers that belong to the same department. Consequently, human resource managers should preferably follow a chaining strategy but carefully ponder whether the cost-benefit ratio for parallel or diverse chains is better.

For the workforce assignment problem, computational tractability was much worse compared to the project selection problem. The standard model was better tractable than the network model, which is less lean, and tractability of the standard model was improved by tight big-M constraints and by globally valid inequalities that are based on lower bounds on the number of required workers for single skill requirements and single projects. Nevertheless, only for small-sized instances with up to 20 projects and 20 workers high-quality solutions could be determined within one hour. Reasons for long computation times and large gaps are quasi-symmetry and an LP relaxation that is—despite its tightening—still weak. Tighter lower bounds that strengthen the LP relaxation would be rewarding and should hence be an area of future research. The heuristics DROP and ROUND would also profit from tighter bounds.

Among the heuristics, DROP(CPLEX, stand.) with (6.22) performed best, followed by ROUND(Tight LP) and ISAP(SSP). ISAP is a very fast heuristic like GRAP, but provides higher solution quality than GRAP. Solving subproblems in ISAP not with a standard LP solver but with a problem-specific method, namely, with a successive shortest path method, drastically accelerated ISAP. However, GRAP and ISAP often fail to return a feasible solution when workforce utilization is high and can make a planner erroneously believe that additional workers are necessary to cover the workload that arises

from projects and departments. When workforce utilization is high, only DROP and ROUND reliably provide feasible solutions. The solution quality of ROUND(Tight LP) is superior to all other heuristics, but solution times are unacceptably high for larger-sized instances. ROUND(Weak LP, (4.18)), on the other hand, is much faster but cannot provide solutions of acceptable quality. These drawbacks of ROUND make DROP the overall best choice. The solution quality of DROP is slightly better than that of ISAP for instances where ISAP can provide feasible solutions. Solving subproblems of DROP with the generalized network simplex method did not pay off. Only for a set of very large-sized instances, DROP(GNS) could provide the best solutions because other DROP variants ran out of memory.

For large firms whose project staffing problems represent large-sized instances of the workforce assignment problem, DROP(CPLEX, stand.) with (6.22) is currently the best method to form small project teams. In our experiments, DROP(CPLEX, stand.) with (6.22) solved instances with up to 1250 workers, 300 projects, 60 skills in total, and 3 skills per worker for a planning horizon that comprised 12 periods. The limited memory access of CPLEX 32 bit hindered DROP to solve instances of larger size. When we used CPLEX 64 bit, solutions could be determined for these instances. Since DROP is embedded in a multi-start approach, solution time and solution quality can be traded off against each other conveniently. To illustrate the trade-off, consider the following results. For instances of the mentioned size with 1250 workers it took 4 minutes to execute 10 passes. 100 passes required half an hour and improved solution quality by 3.5 %.

The heuristic DROP profits from four features. Three of them counteract the drawback of DROP that for each drop operation a large LP must be solved. The first feature is decomposition. Decomposing the feasibility check of a drop operation in feasibility checks for each period means to decompose the large LP into several small LPs; this decomposition reduces checking time considerably. The second feature is the immediate removal of workers who do not contribute to a project from the respective project team. This strategy, which follows an idea of Di Gaspero et al. (2007), reduces the number of feasibility checks and drastically accelerates DROP. Third, lower bounds on the number of needed workers help to reduce the number of required drop operations. Finally, the surrogate objective function (6.22), which is used for the small LPs, steers the drop procedure effectively and improves solution quality.

In our experiments, solution methods exhibited different reactions to changes in workforce utilization and to changes in the degree of multi-skilling. When workforce utilization rose, i.e., when the ratio ρ^{proj} rose, solution methods suffered from increasing solution times to a different degree. The exact branch-and-cut approach of CPLEX and ROUND(Tight LP) suffered seriously. GRAP and ISAP suffered only little but recall that both methods often fail to provide feasible solutions when levels of workforce utilization are high. So actually, GRAP and ISAP suffer most. DROP suffered moderately. Altogether, the computational curse of high workforce utilization was observed for all methods.

With respect to the impact of different degrees of multi-skilling, we can make only preliminary conclusions, because our computational experiments were not as comprehensive as the experiments that we conducted to test the impact of cross-training for the project selection problem. The exact branch-and-cut method of CPLEX and all heuristics except one seem to require more computation time to cope with the complexity of additional

flexibility. The exception is ISAP, which needed less time when workers mastered more skills. Hence, ISAP overcomes the curse of flexibility.

Our approach to the workforce assignment problem has also limitations and offers room for improvements. So far, we have outlined only construction heuristics and these heuristics cannot guarantee to provide high-quality solutions; even for small-sized instances, gaps can be relatively large. Improvement heuristics, e.g., local search methods or evolutionary algorithms, may have the potential to reduce these gaps.

Moreover, our staffing model pursues a single goal, the goal of minimizing average team size. Our literature review, however, has indicated that other aspects can play an important role when teams are formed. Among these aspects is the compatibility of team members, for example. Consequently, it should be worthwhile to include additional objectives into our staffing approach.

Another limitation applies to the project selection and the workforce assignment problem. The models associated with these problems are deterministic models. For both models, we assume that all required data are known and that there is no uncertainty about parameter values. In our opinion, this assumption is justifiable for two reasons. First, we have done fundamental research and it is good practice for basic research to start with the deterministic case. Frequently, insights from a deterministic model can be exploited later when a stochastic model is considered. Furthermore, the allocation of workload in our models can be considered as a rough planning approach. Since our typical period length is one month and since a solution does not prescribe a sequence in which work packages must be accomplished, there is in most cases room for coping with situations where an unexpected increase in skill requirements occurs or where a worker is absent for a day or two. Nevertheless, it would be rewarding to search for robust solutions that guarantee relatively smooth operations even when unexpected changes occur as long as these changes stay within predefined limits.

Despite these limitations, the presented approach can already support human resource managers and other decision makers. All our methods are ready to use; detailed pseudo code is outlined in this work. However, human resource professionals will need support when integrating the methods into their software environment and when adapting the methods to their specific needs. Here, IT professionals and operations research experts are required to put the computational procedures into operation.

Finally, we want to sketch another area where our staffing approach, which aims at small teams, can be applied. So far, we have presumed that our approach is applied by firms where projects are internal improvement projects or customer orders, for example. However, small teams can also be beneficial in other areas. Consider, for example, a rehabilitation facility where the therapy of each patient comprises several treatments. Treatments are provided by medical employees of the facility, e.g., by physicians, nurses, physiotherapists, and other healthcare professionals. Each medical employee is charged to carry out some types of treatments for which he acquired the necessary qualifications. Many treatments such as back massages or special medical gymnastics involve one or more medical employees but only one patient. For reasons of health care quality, it is advantageous when these individual treatments of a patient are carried out by a small and stable team of medical employees and when a medical employee treats a small number of patients frequently and does not treat many patients but each very seldom. Hence, it is advantageous to minimize the number of assignments of medical employees to patients.

Health care quality benefits from small treatment teams because they ease building a familiar relationship between a patient and the treating health care professionals. Additionally, this assignment policy enables a health care professional to pay greater attention to each of her patients and it eases the discussion about the health situation of a patient among those who are involved in caring for this patient.

In this health care setting, our optimization model can be applied to find an assignment of medical employees to patients such that the average number of medical employees that treat a patient is minimized. To transfer our model to the health care setting, consider a patient as a project and assume a period length of one week, for instance. The treatments needed by the patient in each week correspond to the skill requirements of a project. We presume that the treatments needed by a patient have been defined by the physician who admitted the patient to the rehabilitation facility and are thus known in advance. Medical employees correspond to workers. They have different qualifications and each qualification, which is associated with a skill of a worker, allows a medical employee to accomplish a corresponding type of treatments. Here, it may not be necessary or adequate to distinguish different levels of qualifications. A solution of our model returns for each patient a list that states for each required treatment which employees are responsible for the treatment. Upon receipt of the list, the patient can contact these employees in order to make appointments. Because the solution does not prescribe times for treatments, it leaves freedom to the patient and the health care professionals to plan their time also according to their preferences. Note that a solution can split a treatment among medical employees. So, five hours of medical gymnastics that are needed by a patient may be assigned to two physiotherapists. Although such a split may be unfavorable for a single patient, it may be the only way to obtain a feasible assignment when medical staff is highly utilized.

Altogether, our approach to forming small teams makes a novel contribution that closes a long-standing gap. Different areas exist where the model and our solution methods can be applied and where the fruits of this contribution, which appear in form of productive and efficient teamwork, can be reaped.

Chapter 9

Summary and outlook

In this work, we have outlined a hierarchical planning approach to solve three problems that arise in many firms at the interface of multi-project management and human resource management. The problem at the top level is a static project selection problem. The goal of this problem is to determine a project portfolio of maximum benefit that complies with the availabilities and the skills of a given workforce. At the second level, a project team must be composed for each selected project such that all skill requirements of a project can be accomplished by the respective team. The goal of this workforce assignment problem is to minimize average team size. At the bottom level, a utilization leveling problem is considered; after workload of all selected projects has been allocated to workers, departmental workload must be distributed among the workers of each department such that the working times of employees within a department are leveled. Finding good solutions to these problems has great impact on the success of a firm and on the well-being of its employees.

For all three problems, we have formulated mathematical optimization models and analyzed problem complexity. We have proved that the project selection problem and the workforce assignment problem are NP-hard in the strong sense and have shown that the utilization leveling problem can be solved in polynomial time by a specially tailored leveling algorithm. For the project selection problem, we examined the impact that the degree of multi-skilling has on portfolio benefit and found diminishing marginal returns of multi-skilling. Furthermore, we compared different skill configurations for a constant degree of multi-skilling. The tested configurations included alternative chaining strategies. A strategy that features diverse chains performed best; its superiority was verified by established flexibility measures and by a new one. The MIP solver of CPLEX could provide solutions of acceptable quality in reasonable time even for large-sized instances of the project selection problem. For the workforce assignment problem, however, the exact branch-and-cut method of CPLEX performed much worse, even though we tightened the model formulation by globally valid inequalities.

In order to determine solutions for large-sized instances of the workforce assignment problem in acceptable time, we have devised four construction heuristics: a greedy randomized assignment procedure (GRAP), an iterated simultaneous assignment procedure (ISAP), a drop method (DROP), and a rounding heuristic (ROUND). These heuristics have been embedded in a multi-start approach and were thoroughly tested in a comprehensive numerical analysis. For this analysis, an instance generator was devised to create artificial test instances systematically. The heuristic DROP performed best against the test instances and is recommended because it guarantees to find a feasible solution if any exists and because it strikes a good balance between solution time and quality. For

M. Walter, *Multi-Project Management with a Multi-Skilled Workforce,* Produktion und Logistik, DOI 10.1007/978-3-658-08036-5_9, © Springer Fachmedien Wiesbaden 2015

solving subproblems within DROP, we compared the dual simplex method of CPLEX to our implementation of the generalized network simplex method. Here, the LP solver of CPLEX performed better.

Our work has made four novel contributions. First of all, we have provided methods to assemble small project teams of multi-skilled workers with heterogeneous skill levels. The importance of forming small teams has been emphasized in the literature for more than three decades but an approach to this team formation task in a multi-project environment where skill levels of workers are distinguished has been lacking so far. Second, we have outlined detailed pseudo code for implementing the generalized network simplex method. Up to now, textbooks have only sketched the underlying operations.

The other two contributions concern the project selection problem. We analyzed the impact of different skill configurations on portfolio benefit. In particular, we compared a classical skill chaining strategy to an alternative skill chaining strategy, which features diverse chains. Our numerical analysis has shown that the latter strategy is superior. So, our third contribution is the finding that a workforce which exhibits diverse skill chains is highly flexible. The fourth contribution is related to the way how we compared and assessed skill configurations. We used several flexibility measures to assess the quality of skill configurations. One of these measures was newly defined. It is based on a measure of Iravani et al. (2005). Like their measure, our new measure is very lean because it does not require information about the magnitude of skill requirements and about availabilities of workers. However, our new measure has higher discriminatory power than their measure and can serve as a valuable supplementation of their measure.

Especially the workforce assignment problem offers interesting areas for future work. With regard to the tested solution methods, the results of our performance analysis have shown that there is room for improvement. This holds for the exact approach with CPLEX as well as for the heuristics. Our future research will be directed at a genetic algorithm and at an advanced add method. We plan to integrate the genetic algorithm with GRAP or ISAP. Our aim is to combine the speed of these two construction heuristics with the learning strategy of an evolutionary improvement heuristic. For the advanced add method, we plan to follow ideas of Li and Womer (2009a,b) and think of a two-stage decomposition approach. At the first stage, small project teams are formed such that the joint skill set of each team covers all skills required by the corresponding project and such that the number of assignments is leveled across workers. The workload of projects, availabilities of workers, and skill levels are neglected at this stage. At the second stage, a linear program checks whether a feasible allocation of workload exists given the first-stage team assignments. For this check, the objective function of the linear program minimizes the amount of uncovered project workload. If uncovered workload remains, the project with the greatest amount of uncovered workload is identified and for this project an additional worker is demanded in the first-stage problem.

Another avenue for future research emanates from the fact that our hierarchical planning approach has considered only the deterministic case so far. We assumed the absence of uncertainty about data. Though, parameter values that are required for the models can often be estimated only roughly. Then, a robust optimization approach is of interest for risk-adverse decision makers.

A robust approach to our project selection problem should hedge against estimation errors with respect to project benefits and with respect to capacity demand and supply.

If it is not possible to obtain reliable point estimates for the benefit parameters b_p of each project $p \in \mathcal{P}$, the portfolio selection approach of Liesiö et al. (2007), which allows range estimates for parameters, can be a starting point for a modified approach. If skill requirements of projects and availabilities of workers are uncertain, it may be helpful to plan with capacity buffers, e.g., by increasing each skill requirement r_{pst} by a certain percentage.

A robust approach to the workforce assignment problem should at least partially hedge against the absence or departure of a worker. A partial hedge could mean that each skill required by a project is mastered by at least two members of the project team or by even more—depending on the size of the skill requirement.

Multi-objective approaches can be another rewarding area of future work, as we have already indicated in the discussion in Chapter 8. Managers that decide about the project portfolio for the upcoming year may not only seek for a portfolio of maximum benefit but may also wish to level the utilization of the workforce across the months in order to facilitate smooth operations. For the workforce assignment problem, it seems interesting to consider the strategic development of skills of workers as an additional goal, as it has been done by Gutjahr et al. (2008, 2010), Certa et al. (2009), and Süer and Tummaluri (2008), for example. The interesting point is that the goal of a flexible workforce where each worker masters several skills and where the skill levels of each worker are balanced and the goal of small teams have synergies because workers in small teams have greater opportunities to apply all their skills than workers in large teams.

For the inclusion of additional goals into our approach, it may also be beneficial to seize ideas from the works of Lai and Xue (1999), Lopes et al. (2008), Yoshimura et al. (2006), Doerner et al. (2004), Graves and Ringuest (2003), Gutjahr et al. (2008, 2010), Taylor et al. (1982), Valls et al. (2009), Corominas et al. (2005), Eiselt and Marianov (2008), and Certa et al. (2009), whose models feature either multiple objective functions or an objective function that is a weighted sum of multiple objectives. These works can provide inspiration for additional relevant objectives and for suitable solution techniques.

Finally, we want to suggest two further fields of future research. The first field is the integration of scheduling decisions into our hierarchical planning approach. If it is possible to postpone the start times of projects or even of work packages, the outcomes of our approach can be improved by exploiting this degree of freedom. The second field comprises the consideration of more flexible working time regimes that are common in many firms and the consideration of related aspects that are relevant in practice. If, for example, vacation entitlements of workers must be regarded, the concept of partially renewable resources may be helpful for modeling this practice-oriented aspect (cf. Böttcher et al., 1999; Salewski, 1999; Schirmer and Drexl, 2001). Frequently, working time regimes include arrangements that allow for extra hours or for unbalanced monthly hours worked as long as the hours worked during a whole year match the number of contracted hours. In Subsection 4.3.2, we have shown how both arrangements can be represented by constraints in a MIP model. However, in case of the latter arrangement, where hours worked can vary from month to month, only two of our heuristics for the workforce assignment problem, namely, DROP and ROUND, can exploit this flexibility. Though, the temporal decomposition of DROP would no longer be applicable and solution times of DROP would increase considerably. Hence, the development of fast and reliable heuristics that can exploit flexible working times would be another path for future work. Exploiting flex-

ible working times would allow to select even better portfolios and to form even smaller teams.

Bibliography

Aarts, E. H. L. and J. K. Lenstra, eds. (2003): *Local search in combinatorial optimization*. Princeton: Princeton University Press (cit. on p. 124).

Adams, A., E. Lugsden, J. Chase, S. Arber, and S. Bond (2000): Skill-mix changes and work intensification in nursing. *Work, Employment and Society* 14.3, pp. 541–555 (cit. on p. 21).

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin (1993): *Network flows: Theory, algorithms, and applications*. Upper Saddle River: Prentice-Hall (cit. on pp. 61, 62, 74, 75, 158, 161–163, 166–169, 186).

Albrecht, R. and N. Nicol (2007): *Access 2003 programmieren: Professionelle Anwendungsentwicklung mit Access und VBA*. 5th ed. Munich: Addison-Wesley (cit. on p. 197).

Alfares, H. K. and J. E. Bailey (1997): Integrated project task and manpower scheduling. *IIE Transactions* 29.9, pp. 711–717 (cit. on pp. 12–14).

Alfieri, A., P. Brandimarte, and S. D'Orazio (2002): LP-based heuristics for the capacitated lot-sizing problem: The interaction of model formulation and solution algorithm. *International Journal of Production Research* 40.2, pp. 441–458 (cit. on p. 190).

Antoni, C. H. and W. Bungard (2004): Arbeitsgruppen. In: *Enzyklopädie der Psychologie, Themenbereich D: Praxisgebiete, Serie III: Wirtschafts-, Organisations- und Arbeitspsychologie, Bd. 4, Organisationspsychologie – Gruppe und Organisation*. Ed. by H. Schuler. Göttingen: Hogrefe, pp. 129–191 (cit. on pp. 28, 33).

Autor, D. H. and D. Dorn (2013): The growth of low-skill service jobs and the polarization of the US labor market. *American Economic Review* 103.5, pp. 1553–1597 (cit. on p. 18).

Autor, D. H., L. F. Katz, and M. S. Kearney (2006): The polarization of the U.S. labor market. *American Economic Review* 96.2, pp. 189–194 (cit. on p. 18).

Autor, D. H., L. F. Katz, and M. S. Kearney (2008): Trends in U.S. wage inequality: Revising the revisionists. *Review of Economics and Statistics* 90.2, pp. 300–323 (cit. on p. 18).

Balas, E. (1975): Facets of the knapsack polytope. *Mathematical Programming* 8.1, pp. 146–164 (cit. on pp. 112, 113).

Balas, E., S. Ceria, G. Cornuéjols, and N. Natraj (1996): Gomory cuts revisited. *Operations Research Letters* 19.1, pp. 1–9 (cit. on p. 114).

Balinski, M. L. (1965): Integer programming: Methods, uses, computation. *Management Science* 12.3, pp. 253–313 (cit. on p. 113).

Balliet, D. (2010): Communication and cooperation in social dilemmas: A meta-analytic review. *Journal of Conflict Resolution* 54.1, pp. 39–57 (cit. on p. 34).

M. Walter, *Multi-Project Management with a Multi-Skilled Workforce,* Produktion und Logistik, DOI 10.1007/978-3-658-08036-5, © Springer Fachmedien Wiesbaden 2015

Banker, R. D., J. M. Field, R. G. Schroeder, and K. K. Sinha (1996): Impact of work teams on manufacturing performance: A longitudinal field study. *Academy of Management Journal* 39.4, pp. 867–890 (cit. on p. 29).

Barr, R. S., F. Glover, and D. Klingman (1977): The alternating basis algorithm for assignment problems. *Mathematical Programming* 13.1, pp. 1–13 (cit. on p. 146).

Barreto, A., M. d. O. Barros, and C. M. L. Werner (2008): Staffing a software project: A constraint satisfaction and optimization-based approach. *Computers & Operations Research* 35.10, pp. 3073–3089 (cit. on pp. 12–15, 17).

Bassett, M. (2000): Assigning projects to optimize the utilization of employees' time and expertise. *Computers & Chemical Engineering* 24.2-7, pp. 1013–1021 (cit. on pp. 13, 14).

Bazaraa, M. S., J. J. Jarvis, and H. D. Sherali (2010): *Linear programming and network flows*. 4th ed. Hoboken: Wiley (cit. on pp. 161, 173–175, 179).

Beck, S. (2003): Skill and competence management as a base of an integrated personnel development (IPD)—A pilot project in the Putzmeister, Inc./Germany. *Journal of Universal Computer Science* 9.12, pp. 1381–1387 (cit. on p. 20).

Behrens, C.-U. and M. Kirspel (2003): *Grundlagen der Volkswirtschaftslehre*. 3rd ed. Munich: Oldenbourg (cit. on p. 139).

Bellenguez-Morineau, O. and E. Néron (2007): A branch-and-bound method for solving multi-skill project scheduling problem. *RAIRO Operations Research* 41.2, pp. 155–170 (cit. on pp. 12, 13).

Benoist, T. (2007): Towards optimal formwork pairing on construction sites. *RAIRO Operations Research* 41.4, pp. 381–398 (cit. on p. 84).

Benoist, T. and F. Chauvet (2001): *Complexity of some FPP related problems*. e-lab Research Report, Bouygues SA, Paris (cit. on pp. 83–85).

Berman, E., J. Bound, and S. Machin (1998): Implications of skill-biased technological change: International evidence. *Quarterly Journal of Economics* 113.4, pp. 1245–1279 (cit. on p. 18).

Berrada, M. and K. E. Stecke (1986): A branch and bound approach for machine load balancing in flexible manufacturing systems. *Management Science* 32.10, pp. 1316–1335 (cit. on p. 68).

Bertsekas, D. (1998): *Network optimization: Continuous and discrete models*. Belmont: Athena Scientific (cit. on p. 61).

Bertsimas, D. and R. Vohra (1998): Rounding algorithms for covering problems. *Mathematical Programming* 80.1, pp. 63–89 (cit. on p. 190).

Birge, J. R. and F. Louveaux (1997): *Introduction to stochastic programming*. New York: Springer (cit. on p. 25).

Bixby, R. and E. Rothberg (2007): Progress in computational mixed integer programming—A look back from the other side of the tipping point. *Annals of Operations Research* 149.1, pp. 37–41 (cit. on p. 110).

Bloemhof-Ruwaard, J. M., M. Salomon, and L. N. Van Wassenhove (1996): The capacitated distribution and waste disposal problem. *European Journal of Operational Research* 88.3, pp. 490–503 (cit. on p. 190).

BMWi (2014): *Fachkräfteengpässe in Unternehmen*. Ed. by Bundesministerium für Wirtschaft und Energie. Berlin. URL: `http://www.bmwi.de/DE/Mediathek/publikationen,did=623884.html` (visited on Apr. 9, 2014) (cit. on pp. 1, 18).

Bofinger, P. (2011): *Grundzüge der Volkswirtschaftslehre*. 3rd ed. Munich: Pearson Studium (cit. on p. 139).

Boh, W. F., S. A. Slaughter, and J. A. Espinosa (2007): Learning from experience in software development: A multilevel analysis. *Management Science* 53.8, pp. 1315–1331 (cit. on p. 66).

Bonatti, A. and J. Hörner (2011): Collaborating. *American Economic Review* 101.2, pp. 632–663 (cit. on p. 32).

Borjas, G. J. (2013): *Labor economics*. 6th ed. New York: McGraw-Hill (cit. on pp. 1, 18).

Bosch, R. and M. Trick (2005): Integer Programming. In: *Search methodologies*. Ed. by E. K. Burke and G. Kendall. New York: Springer, pp. 69–95 (cit. on pp. 59, 232).

Böttcher, J., A. Drexl, R. Kolisch, and F. Salewski (1999): Project scheduling under partially renewable resource constraints. *Management Science* 45.4, pp. 543–559 (cit. on p. 283).

Brooks, F. P. (1982): *The mythical man-month: Essays on software engineering*. Reading: Addison-Wesley (cit. on pp. 3, 35, 92).

Brown, G. G. and R. D. McBride (1984): Solving generalized networks. *Management Science* 30.12, pp. 1497–1523 (cit. on p. 186).

Brucker, P. and L. Nordmann (1994): The k-track assignment problem. *Computing* 52.2, pp. 97–122 (cit. on p. 93).

Brusco, M. J. and T. R. Johns (1998): Staffing a multiskilled workforce with varying levels of productivity: An analysis of cross-training policies. *Decision Sciences* 29.2, pp. 499–515 (cit. on pp. 15, 16, 23, 96).

Burgess, A. R. and J. B. Killebrew (1962): Variation in activity level on a cyclical arrow diagram. *Journal of Industrial Engineering* 13.2, pp. 76–83 (cit. on p. 68).

Burkard, R., M. Dell'Amico, and S. Martello (2009): *Assignment problems*. Philadelphia: SIAM (Society for Industrial and Applied Mathematics) (cit. on pp. 82, 145, 146).

Burke, E. K. and G. Kendall, eds. (2005): *Search methodologies*. New York: Springer (cit. on p. 124).

Campbell, G. M. (1999): Cross-utilization of workers whose capabilities differ. *Management Science* 45.5, pp. 722–732 (cit. on pp. 21, 23, 24, 276).

Campbell, G. M. and M. Diaby (2002): Development and evaluation of an assignment heuristic for allocating cross-trained workers. *European Journal of Operational Research* 138.1, pp. 9–20 (cit. on pp. 15, 16, 21).

Cass, D. J. (1992): Labor productivity impact of varying crew levels. *Transactions of the American Association of Cost Engineers*, pp. C.2.1–C.2.9 (cit. on p. 35).

Cassera, M. A., B. Zheng, D. V. Martinec, C. M. Dunst, and L. L. Swanström (2009): Surgical time independently affected by surgical team size. *American Journal of Surgery* 198.2, pp. 216–222 (cit. on p. 34).

Certa, A., M. Enea, G. Galante, and C. M. La Fata (2009): Multi-objective human resources allocation in R&D projects planning. *International Journal of Production Research* 47.13, pp. 3503–3523 (cit. on pp. 15, 16, 98, 99, 283).

Chang, M. D., M. Engquist, R. Finkel, and R. R. Meyer (1988): A parallel algorithm for generalized networks. *Annals of Operations Research* 14.1, pp. 125–145 (cit. on p. 186).

Chen, A. N. K. and T. M. Edgington (2005): Assessing value in organizational knowledge creation: Considerations for knowledge workers. *MIS Quarterly* 29.2, pp. 279–309 (cit. on p. 65).

Chen, J. and R. G. Askin (2009): Project selection, scheduling and resource allocation with time dependent returns. *European Journal of Operational Research* 193.1, pp. 23–34 (cit. on pp. 9, 10, 13, 14).

Chou, M. C., G. A. Chua, C.-P. Teo, and H. Zheng (2011): Process flexibility revisited: The graph expander and its applications. *Operations Research* 59.5, pp. 1090–1105 (cit. on pp. 2, 25–27, 222, 223, 276).

Chvátal, V. (1983): *Linear programming*. New York: Freeman (cit. on pp. 110, 121, 183, 255).

Clark, R. H., J. L. Kennington, R. R. Meyer, and M. Ramamurti (1992): Generalized networks: Parallel algorithms and an empirical analysis. *ORSA/INFORMS Journal on Computing* 4.2, pp. 132–145 (cit. on p. 186).

Comer, D. R. (1995): A model of social loafing in real work groups. *Human Relations* 48.6, pp. 647–667 (cit. on p. 32).

Conforti, M., G. Cornuéjols, and G. Zambelli (2010): Polyhedral approaches to mixed integer linear programming. In: *50 years of integer programming 1958-2008*. Ed. by M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey. Heidelberg: Springer, pp. 343–386 (cit. on p. 113).

Cordeau, J.-F., G. Laporte, F. Pasin, and S. Ropke (2010): Scheduling technicians and tasks in a telecommunications company. *Journal of Scheduling* 13.4, pp. 393–409 (cit. on pp. 13, 14).

Cornuéjols, G., R. Sridharan, and J. M. Thizy (1991): A comparison of heuristics and relaxations for the capacitated plant location problem. *European Journal of Operational Research* 50.3, pp. 280–297 (cit. on p. 149).

Corominas, A., J. Ojeda, and R. Pastor (2005): Multi-objective allocation of multi-function workers with lower bounded capacity. *Journal of the Operational Research Society* 56.6, pp. 738–743 (cit. on pp. 15, 283).

Correia, I., L. Lourenço, and F. Saldanha-da-Gama (2012): Project scheduling with flexible resources: Formulation and inequalities. *OR Spectrum* 34.3, pp. 635–663 (cit. on p. 13).

Cruz, F. R. B., J. M. Smith, and G. R. Mateus (1998): Solving to optimality the uncapacitated fixed-charge network flow problem. *Computers & Operations Research* 25.1, pp. 67–81 (cit. on p. 83).

Cunningham, W. H. (1979): Theoretical properties of the network simplex method. *Mathematics of Operations Research* 4.2, pp. 196–208 (cit. on p. 183).

Dantzig, G. B. (1963): *Linear programming and extensions*. Princeton: Princeton University Press (cit. on pp. 110, 161).

Dantzig, G. B. and M. N. Thapa (1997): *Linear programming 1: Introduction*. New York: Springer (cit. on p. 161).

Davis, J. H. (1969): *Group performance*. Reading: Addison-Wesley (cit. on pp. 31, 33).

De Reyck, B. and W. Herroelen (1999): The multi-mode resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research* 119.2, pp. 538–556 (cit. on p. 11).

Delarue, A., G. Van Hootegem, S. Procter, and M. Burridge (2008): Teamworking and organizational performance: A review of survey-based research. *International Journal of Management Reviews* 10.2, pp. 127–148 (cit. on p. 28).

Demary, M. and V. Erdmann (2012): Fachkräfteengpässe und Arbeitslosigkeit in Europa – Wanderung als kurzfristiger Ausgleichmechanismus. *IW-Trends* 39.3, pp. 35–48 (cit. on p. 18).

Di Gaspero, L., J. Gärtner, G. Kortsarz, N. Musliu, A. Schaerf, and W. Slany (2007): The minimum shift design problem. *Annals of Operations Research* 155.1, pp. 79–105 (cit. on pp. 154, 277).

Dillenberger, C., L. F. Escudero, A. Wollensak, and W. Zhang (1994): On practical resource allocation for production planning and scheduling with period overlapping setups. *European Journal of Operational Research* 75.2, pp. 275–286 (cit. on p. 195).

Dodin, B. and A. A. Elimam (1997): Audit scheduling with overlapping activities and sequence-dependent setup costs. *European Journal of Operational Research* 97.1, pp. 22–33 (cit. on pp. 12, 13).

Doerner, K., W. Gutjahr, R. Hartl, C. Strauss, and C. Stummer (2004): Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection. *Annals of Operations Research* 131.1, pp. 79–99 (cit. on pp. 10, 88, 283).

Domschke, W. and A. Drexl (1996): *Logistik: Standorte*. 4th ed. Munich: Oldenbourg (cit. on p. 149).

Domschke, W. and A. Drexl (2007): *Einführung in Operations Research*. 7th ed. Berlin: Springer (cit. on pp. 54, 124).

Drexl, A. (1991): Scheduling of project networks by job assignment. *Management Science* 37.12, pp. 1590–1602 (cit. on p. 13).

Drezet, L.-E. and J.-C. Billaut (2008): A project scheduling problem with labour constraints and time-dependent activities requirements. *International Journal of Production Economics* 112.1, pp. 217–225 (cit. on p. 13).

Dustmann, C., J. Ludsteck, and U. Schönberg (2009): Revisiting the German wage structure. *Quarterly Journal of Economics* 124.2, pp. 843–881 (cit. on p. 18).

Earley, P. C. (1989): Social loafing and collectivism: A comparison of the United States and the People's Republic of China. *Administrative Science Quarterly* 34.4, pp. 565–581 (cit. on p. 32).

Earley, P. C. (1993): East meets West meets Mideast: Further explorations of collectivistic and individualistic work groups. *Academy of Management Journal* 36.2, pp. 319–348 (cit. on p. 32).

Easa, S. M. (1989): Resource leveling in construction by optimization. *Journal of Construction Engineering and Management* 115.2, pp. 302–316 (cit. on p. 68).

Edmonds, J. (1965): Paths, trees, and flowers. *Canadian Journal of Mathematics* 17, pp. 449–467 (cit. on p. 82).

Ehrgott, M. (2005): *Multicriteria optimization*. 2nd ed. Berlin: Springer (cit. on p. 54).

Eilat, H., B. Golany, and A. Shtub (2006): Constructing and evaluating balanced portfolios of R&D projects with interactions: A DEA based methodology. *European Journal of Operational Research* 172.3, pp. 1018–1039 (cit. on pp. 10, 45).

Eiselt, H. A. and V. Marianov (2008): Employee positioning and workload allocation. *Computers & Operations Research* 35.2, pp. 513–524 (cit. on pp. 15, 16, 49, 106, 107, 283).

Elam, J., F. Glover, and D. Klingman (1979): A strongly convergent primal simplex algorithm for generalized networks. *Mathematics of Operations Research* 4.1, pp. 39–59 (cit. on p. 183).

Elmuti, D. (1997): The perceived impact of team-based management systems on organizational effectiveness. *Team Performance Management* 3.3, pp. 179–192 (cit. on p. 29).

Engquist, M. (1982): A successive shortest path algorithm for the assignment problem. *INFOR* 20.4, pp. 370–384 (cit. on p. 146).

Erpenbeck, J. and L. v. Rosenstiel (2007): Einführung. In: *Handbuch Kompetenzmessung*. Ed. by J. Erpenbeck and L. v. Rosenstiel. Stuttgart: Schäffer-Poeschel, pp. XVII–XLVI (cit. on pp. 19, 20).

Escudero, L. F. and J. Salmeron (2005): On a fix-and-relax framework for a class of project scheduling problems. *Annals of Operations Research* 140.1, pp. 163–188 (cit. on pp. 9, 10, 13, 14, 88, 195).

Feldman, E., F. A. Lehrer, and T. L. Ray (1966): Warehouse location under continuous economies of scale. *Management Science* 12.9, pp. 670–684 (cit. on p. 149).

Filley, A. C., R. J. House, and S. Kerr (1976): *Managerial process and organizational behavior*. 2nd ed. Glenview: Scott, Foresman and Company (cit. on p. 30).

Fitzpatrick, E. L. and R. G. Askin (2005): Forming effective worker teams with multi-functional skill requirements. *Computers & Industrial Engineering* 48.3, pp. 593–608 (cit. on pp. 15, 16).

Fırat, M. and C. A. J. Hurkens (2012): An improved MIP-based approach for a multi-skill workforce scheduling problem. *Journal of Scheduling* 15.3, pp. 363–380 (cit. on pp. 13, 14).

Flanagan, J. C. (1954): The critical incident technique. *Psychological Bulletin* 51.4, pp. 327–358 (cit. on p. 19).

Förster, A., K. Haase, and M. Tönnies (2006): Ein modellgestützter Ansatz zur mittelfristigen Produktions- und Ablaufplanung für eine Brauerei. *Zeitschrift für Betriebswirtschaft* 76.12, pp. 1255–1274 (cit. on p. 195).

Fowler, J. W., P. Wirojanagud, and E. S. Gel (2008): Heuristics for workforce planning with worker differences. *European Journal of Operational Research* 190.3, pp. 724–740 (cit. on pp. 15, 16, 190).

Fox, G. E., N. R. Baker, and J. L. Bryant (1984): Economic models for R and D project selection in the presence of project interactions. *Management Science* 30.7, pp. 890–902 (cit. on pp. 10, 44).

Gabrenya, W. K., Y.-E. Wang, and B. Latané (1985): Social loafing on an optimizing task: Cross-cultural differences among Chinese and Americans. *Journal of Cross-Cultural Psychology* 16.2, pp. 223–242 (cit. on p. 32).

GAMS Development Corporation (2013): *GAMS—A User's Guide, GAMS Version 24 Release 1.2*. GAMS Development Corporation, Washington. URL: `http://www.gams.com/dd/docs/bigdocs/GAMSUsersGuide.pdf` (visited on June 4, 2013) (cit. on p. 197).

Garey, M. R. and D. S. Johnson (1979): *Computers and intractability: A guide to the theory of NP-completeness*. New York: Freeman (cit. on pp. 25, 74–80, 83–85, 109).

Garey, M., D. Johnson, G. Miller, and C. Papadimitriou (1980): The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic and Discrete Methods* 1.2, pp. 216–227 (cit. on p. 93).

Gather, T. (2011): *Exakte Verfahren für das Ressourcennivellierungsproblem*. Wiesbaden: Gabler (cit. on p. 104).

George, J. M. (1992): Extrinsic and intrinsic origins of perceived social loafing in organizations. *Academy of Management Journal* 35.1, pp. 191–202 (cit. on p. 33).

Ghasemzadeh, F., N. Archer, and P. Iyogun (1999): A zero-one model for project portfolio selection and scheduling. *Journal of the Operational Research Society* 50.7, pp. 745–755 (cit. on pp. 9, 10, 13, 14, 45, 88).

Gicquel, C., N. Miégeville, M. Minoux, and Y. Dallery (2010): Optimizing glass coating lines: MIP model and valid inequalities. *European Journal of Operational Research* 202.3, pp. 747–755 (cit. on p. 110).

Glover, F., R. Glover, and D. Klingman (1986): Threshold assignment algorithm. In: *Netflow at Pisa*. Ed. by G. Gallo and C. Sandi. Mathematical Programming Studies 26. Berlin: Springer, pp. 12–37 (cit. on pp. 146, 147, 249).

Glover, F. and E. Woolsey (1974): Converting the 0–1 polynomial programming problem to a 0–1 linear program. *Operations Research* 22.1, pp. 180–182 (cit. on p. 45).

Gnahs, D. (2010): *Kompetenzen – Erwerb, Erfassung, Instrumente*. Bielefeld: Bertelsmann (cit. on pp. 19, 20).

Golabi, K., C. W. Kirkwood, and A. Sicherman (1981): Selecting a portfolio of solar energy projects using multiattribute preference theory. *Management Science* 27.2, pp. 174–189 (cit. on p. 10).

Goldfarb, D. and M. J. Todd (1989): Linear programming. In: *Optimization*. Ed. by G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd. Handbooks in Operations Research and Management Science 1. Amsterdam: Elsevier, pp. 73–170 (cit. on pp. 121, 123).

Gomar, J. E., C. T. Haas, and D. P. Morton (2002): Assignment and allocation optimization of partially multiskilled workforce. *Journal of Construction Engineering and Management* 128.2, pp. 103–109 (cit. on pp. 15, 16, 24, 96, 97, 276).

Gomes, C. P. and R. Williams (2005): Approximation algorithms. In: *Search methodologies*. Ed. by E. K. Burke and G. Kendall. New York: Springer, pp. 557–585 (cit. on p. 109).

Gomory, R. E. (1958): Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society* 64.5, pp. 275–278 (cit. on p. 113).

Gomory, R. E. (1960): *An algorithm for the mixed integer problem*. Research Memorandum RM-2597-PR, RAND Corporation, Santa Monica (cit. on p. 113).

Goos, M., A. Manning, and A. Salomons (2009): Job polarization in Europe. *American Economic Review* 99.2, pp. 58–63 (cit. on p. 18).

GPM (2008): *ProjektManager*. Ed. by GPM Deutsche Gesellschaft für Projektmanagement e.V., H. Schelle, R. Ottmann, and A. Pfeiffer. 3rd ed. Nürnberg: GPM (cit. on p. 9).

Graves, S. B. and J. L. Ringuest (2003): *Models & methods for project selection: Concepts from management science, finance, and information technology*. Boston: Kluwer Academic Publishers (cit. on pp. 10, 11, 87, 283).

Green, F. (2004): Work intensification, discretion, and the decline in well-being at work. *Eastern Economic Journal* 30.4, pp. 615–625 (cit. on p. 21).

Grunow, M., H.-O. Günther, and G. Yang (2004): Development of a decision support model for scheduling clinical studies and assigning medical personnel. *Health Care Management Science* 7.4, pp. 305–317 (cit. on pp. 3, 13, 14, 54, 57, 93, 99–103, 275).

Grunow, M., H.-O. Günther, and M. Lehmann (2002): Campaign planning for multi-stage batch processes in the chemical industry. *OR Spectrum* 24.3, pp. 281–314 (cit. on p. 57).

Guerrero, F., S. Lozano, T. Koltai, and J. Larrañeta (1999): Machine loading and part type selection in flexible manufacturing systems. *International Journal of Production Research* 37.6, pp. 1303–1317 (cit. on p. 68).

Guisewite, G. and P. M. Pardalos (1990): Minimum concave-cost network flow problems: Applications, complexity, and algorithms. *Annals of Operations Research* 25.1, pp. 75–99 (cit. on pp. 83, 85).

Günther, H.-O. (1989): *Produktionsplanung bei flexibler Personalkapazität*. Stuttgart: Poeschel (cit. on p. 54).

Gupta, U. I., D. T. Lee, and J. Y.-T. Leung (1979): An optimal solution for the channel-assignment problem. *IEEE Transactions on Computers* C-28.11, pp. 807–810 (cit. on p. 93).

Gurgur, C. Z. and C. T. Morley (2008): Lockheed Martin Space Systems Company optimizes infrastructure project-portfolio selection. *Interfaces* 38.4, pp. 251–262 (cit. on p. 10).

Gutjahr, W. J., S. Katzensteiner, P. Reiter, C. Stummer, and M. Denk (2008): Competence-driven project portfolio selection, scheduling and staff assignment. *Central European Journal of Operations Research* 16.3, pp. 281–306 (cit. on pp. 2, 10, 12–14, 65, 90–92, 128, 283).

Gutjahr, W. J., S. Katzensteiner, P. Reiter, C. Stummer, and M. Denk (2010): Multi-objective decision analysis for competence-oriented project portfolio selection. *European Journal of Operational Research* 205.3, pp. 670–679 (cit. on pp. 10–14, 65, 92, 283).

Hackman, J. R. (1987): The design of work teams. In: *Handbook of organizational behavior*. Ed. by J. W. Lorsch. Englewood Cliffs: Prentice-Hall, pp. 315–342 (cit. on p. 30).

Hamburger, H., M. Guyer, and J. Fox (1975): Group size and cooperation. *Journal of Conflict Resolution* 19.3, pp. 503–531 (cit. on p. 32).

Hammer, M. and J. Champy (1993): *Reengineering the corporation: A manifesto for business revolution*. New York: Harper Business (cit. on pp. 3, 35).

Hanna, A. S., C.-K. Chang, J. A. Lackney, and K. T. Sullivan (2007): Impact of overmanning on mechanical and sheet metal labor productivity. *Journal of Construction Engineering and Management* 133.1, pp. 22–28 (cit. on p. 36).

He, J. (2012): Counteracting free-riding with team morale—An experimental study. *Project Management Journal* 43.3, pp. 62–75 (cit. on p. 33).

Hegazy, T., A. Shabeeb, E. Elbeltagi, and T. Cheema (2000): Algorithm for scheduling with multiskilled constrained resources. *Journal of Construction Engineering and Management* 126.6, pp. 414–421 (cit. on pp. 13, 14).

Heidenberger, K. and C. Stummer (1999): Research and development project selection and resource allocation: A review of quantitative modelling approaches. *International Journal of Management Reviews* 1.2, pp. 197–224 (cit. on p. 87).

Heilmann, R. (2003): A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags. *European Journal of Operational Research* 144.2, pp. 348–365 (cit. on p. 11).

Heimerl, C. and R. Kolisch (2010a): Scheduling and staffing multiple projects with a multi-skilled workforce. *OR Spectrum* 32.2, pp. 369–394 (cit. on pp. 3, 4, 7, 13, 24, 42, 64, 93, 99, 102, 224, 276).

Heimerl, C. and R. Kolisch (2010b): Work assignment to and qualification of multi-skilled human resources under knowledge depreciation and company skill level targets. *International Journal of Production Research* 48.13, pp. 3759–3781 (cit. on p. 65).

Hendriks, M. H. A., B. Voeten, and L. Kroep (1999): Human resource allocation in a multi-project R&D environment: Resource capacity allocation and project portfolio planning in practice. *International Journal of Project Management* 17.3, pp. 181–188 (cit. on pp. 3, 36).

Herbots, J., W. Herroelen, and R. Leus (2007): Dynamic order acceptance and capacity planning on a single bottleneck resource. *Naval Research Logistics* 54.8, pp. 874–889 (cit. on p. 8).

Hertel, G. (2011): Synergetic effects in working teams. *Journal of Managerial Psychology* 26.3, pp. 176–184 (cit. on p. 28).

Hiermann, W. and M. Höfferer (2005): Skill management: Searching highly skilled employees for teambuilding and project management tasks. In: *Proceedings of the 5th International Conference on Knowledge Management and Knowledge Technologies (i-KNOW)*, pp. 396–403. URL: `http://knowminer.know-center.tugraz.at/corpi/iknow-papers/papers/pdf/6_Skill%20Management.pdf` (visited on Apr. 9, 2014) (cit. on p. 20).

Hill, R. E. (1975): Interpersonal compatibility and workgroup performance. *Journal of Applied Behavioral Science* 11.2, pp. 210–219 (cit. on p. 67).

Ho, S. C. and J. M. Y. Leung (2010): Solving a manpower scheduling problem for airline catering using metaheuristics. *European Journal of Operational Research* 202.3, pp. 903–921 (cit. on pp. 13, 14, 105).

Högl, M. (1998): *Teamarbeit in innovativen Projekten: Einflussgrößen und Wirkungen*. Wiesbaden: Gabler (cit. on pp. 29, 35, 36).

Huang, F.-H. and S.-L. Hwang (2009): Experimental studies of computerized procedures and team size in nuclear power plant operations. *Nuclear Engineering and Design* 239.2, pp. 373–380 (cit. on p. 36).

Huang, H. C., C. Lee, and Z. Xu (2006): The workload balancing problem at air cargo terminals. *OR Spectrum* 28.4, pp. 705–727 (cit. on pp. 4, 49, 105).

IBM (2011): *IBM ILOG CPLEX Optimization Studio Version 12 Release 4: CPLEX User's Manual*. IBM, Armonk. URL: `http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r4/topic/ilog.odms.studio.help/pdf/usrcplex.pdf` (visited on Apr. 9, 2014) (cit. on p. 197).

Ingham, A. G., G. Levinger, J. Graves, and V. Peckham (1974): The Ringelmann effect: Studies of group size and group performance. *Journal of Experimental Social Psychology* 10.4, pp. 371–384 (cit. on p. 31).

Ingram, T. N., R. W. LaForge, and T. W. Leigh (2002): Selling in the new millennium: A joint agenda. *Industrial Marketing Management* 31.7, pp. 559–567 (cit. on p. 34).

Inman, R. R., D. E. Blumenfeld, and A. Ko (2005): Cross-training hospital nurses to reduce staffing costs. *Health Care Management Review* 30.2, pp. 116–125 (cit. on pp. 20, 21).

Iravani, S. M., M. P. Van Oyen, and K. T. Sims (2005): Structural flexibility: A new perspective on the design of manufacturing and service operations. *Management Science* 51.2, pp. 151–166 (cit. on pp. 2, 25–27, 222, 223, 282).

Jacobsen, S. K. (1983): Heuristics for the capacitated plant location model. *European Journal of Operational Research* 12.3, pp. 253–261 (cit. on p. 149).

Jang, S. Y., J. Park, and N. Park (1996): An integrated decision support system for FMS production planning and scheduling problems. *International Journal of Advanced Manufacturing Technology* 11.2, pp. 101–110 (cit. on p. 68).

Jensen, P. A. and J. W. Barnes (1987): *Network flow programming*. 2nd ed. Malabar: Krieger (cit. on p. 161).

Jones, R. W. (1961): Comparative advantage and the theory of tariffs: A multi-country, multi-commodity model. *Review of Economic Studies* 28.3, pp. 161–175 (cit. on p. 140).

Jordan, W. C. and S. C. Graves (1995): Principles on the benefits of manufacturing process flexibility. *Management Science* 41.4, pp. 577–594 (cit. on pp. 2, 21–23, 276).

Josuttis, N. M. (2001): *Objektorientiertes Programmieren in C++: Ein Tutorial für Ein- und Umsteiger*. 2nd ed. Munich: Addison-Wesley (cit. on p. 197).

Josuttis, N. M. (2002): *Object-oriented programming in C++*. Chichester: Wiley (cit. on p. 197).

Jungnickel, D. (2005): *Graphs, networks and algorithms*. 2nd ed. Berlin: Springer (cit. on pp. 82, 161).

Kaparis, K. and A. N. Letchford (2008): Local and global lifted cover inequalities for the 0–1 multidimensional knapsack problem. *European Journal of Operational Research* 186.1, pp. 91–103 (cit. on p. 81).

Karau, S. J. and K. D. Williams (1993): Social loafing: A meta-analytic review and theoretical integration. *Journal of Personality and Social Psychology* 65.4, pp. 681–706 (cit. on pp. 31, 32).

Kellerer, H., U. Pferschy, and D. Pisinger (2004): *Knapsack problems*. Berlin: Springer (cit. on pp. 80, 81).

Kerr, N. L. (1983): Motivation losses in small groups: A social dilemma analysis. *Journal of Personality and Social Psychology* 45.4, pp. 819–828 (cit. on p. 31).

Kerzner, H. (2013): *Project management: A systems approach to planning, scheduling, and controlling*. 11th ed. Hoboken: Wiley (cit. on pp. 1, 7, 42, 66).

Kim, D. and P. M. Pardalos (1999): A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure. *Operations Research Letters* 24.4, pp. 195–203 (cit. on p. 83).

Kimms, A. and M. Müller-Bungart (2007): Revenue management for broadcasting commercials: The channel's problem of selecting and scheduling the advertisements to be aired. *International Journal of Revenue Management* 1.1, pp. 28–44 (cit. on pp. 190, 196).

Klimoski, R. and R. G. Jones (1995): Staffing for effective group decision making: Key issues in matching people and teams. In: *Team effectiveness and decision making in organizations*. Ed. by R. A. Guzzo and E. Salas. San Francisco: Jossey-Bass, pp. 291–332 (cit. on pp. 29, 30).

Kolen, A. W. J., J. K. Lenstra, C. H. Papadimitriou, and F. C. R. Spieksma (2007): Interval scheduling: A survey. *Naval Research Logistics* 54.5, pp. 530–543 (cit. on pp. 11, 93, 101).

Kolisch, R. and C. Heimerl (2012): An efficient metaheuristic for integrated scheduling and staffing IT projects based on a generalized minimum cost flow network. *Naval Research Logistics* 59.2, pp. 111–127 (cit. on pp. 12, 13, 102, 103, 160, 255).

Kolisch, R., C. Heimerl, and S. Hausen (2008): Projektportfolio- und Multiprojektplanung: Modellierung, prototypische Implementierung und Einsatz in der Finanzdienstleistungsbranche. *Zeitschrift für Betriebswirtschaft* 78.6, pp. 591–610 (cit. on pp. 9, 10, 12–14, 88).

Kolisch, R., K. Meyer, and R. Mohr (2005): Maximizing R&D portfolio value. *Research-Technology Management* 48.3, pp. 33–39 (cit. on pp. 2, 9, 10).

Krishnamoorthy, M., A. T. Ernst, and D. Baatar (2012): Algorithms for large scale shift minimisation personnel task scheduling problems. *European Journal of Operational Research* 219.1, pp. 34–48 (cit. on pp. 15, 16, 95, 96).

Kroon, L. G., M. Salomon, and L. N. Van Wassenhove (1997): Exact and approximation algorithms for the tactical fixed interval scheduling problem. *Operations Research* 45.4, pp. 624–638 (cit. on p. 94).

Krugman, P. R., M. Obstfeld, and M. J. Melitz (2012): *International economics: Theory & policy*. 9th ed. Boston: Pearson Education (cit. on p. 139).

Kuehn, A. A. and M. J. Hamburger (1963): A heuristic program for locating warehouses. *Management Science* 9.4, pp. 643–666 (cit. on p. 149).

Kumar, A., R. Dijkman, and M. Song (2013): Optimal resource assignment in workflows for maximizing cooperation. In: *Business Process Management: Proceedings of the 11th International Conference on Business Process Management*. Ed. by F. Daniel, J. Wang, and B. Weber. Berlin: Springer, pp. 235–250 (cit. on pp. 15, 16, 67).

Kumar, N. and K. Shanker (2001): Comparing the effectiveness of workload balancing objectives in FMS loading. *International Journal of Production Research* 39.5, pp. 843–871 (cit. on pp. 68, 104).

Lai, K. K. and J. Xue (1999): Decision modelling of diversified project selection. *Annals of Operations Research* 87, pp. 199–212 (cit. on pp. 10, 44, 45, 88, 283).

Latané, B., K. Williams, and S. Harkins (1979): Many hands make light the work: The causes and consequences of social loafing. *Journal of Personality and Social Psychology* 37.6, pp. 822–832 (cit. on p. 31).

Laux, H. and F. Liermann (2003): *Grundlagen der Organisation*. 5th ed. Berlin: Springer (cit. on p. 40).

LeBlanc, L. J., D. J. Randels, and T. K. Swann (2000): Heery International's spreadsheet optimization model for assigning managers to construction projects. *Interfaces* 30.6, pp. 95–106 (cit. on pp. 15, 16, 106).

Lee, T.-R. and J.-H. Ueng (1999): A study of vehicle routing problems with load-balancing. *International Journal of Physical Distribution & Logistics Management* 29.10, pp. 646–657 (cit. on pp. 4, 49, 103, 104).

Levine, E. L., R. A. Ash, H. Hall, and F. Sistrunk (1983): Evaluation of job analysis methods by experienced job analysts. *Academy of Management Journal* 26.2, pp. 339–348 (cit. on p. 19).

Levine, E. L., F. Sistrunk, K. J. McNutt, and S. Gael (1988): Exemplary job analysis systems in selected organizations: A description of process and outcomes. *Journal of Business and Psychology* 3.1, pp. 3–21 (cit. on p. 19).

Li, H. and K. Womer (2009a): A decomposition approach for shipboard manpower scheduling. *Military Operations Research* 14.3, pp. 67–90 (cit. on pp. 12, 13, 93, 97, 98, 282).

Li, H. and K. Womer (2009b): Scheduling projects with multi-skilled personnel by a hybrid MILP/CP benders decomposition algorithm. *Journal of Scheduling* 12.3, pp. 281–298 (cit. on pp. 12, 13, 93, 97, 98, 282).

Liden, R. C., S. J. Wayne, R. A. Jaworski, and N. Bennett (2004): Social loafing: A field investigation. *Journal of Management* 30.2, pp. 285–304 (cit. on pp. 3, 33).

Liesiö, J., P. Mild, and A. Salo (2007): Preference programming for robust portfolio modeling and project selection. *European Journal of Operational Research* 181.3, pp. 1488–1505 (cit. on pp. 44, 283).

Lopes, L., M. Aronson, G. Carstensen, and C. Smith (2008): Optimization support for senior design project assignments. *Interfaces* 38.6, pp. 448–464 (cit. on pp. 10, 11, 15, 16, 283).

Lopez, F. M., G. A. Kesselman, and F. E. Lopez (1981): An empirical test of a trait-oriented job analysis technique. *Personnel Psychology* 34.3, pp. 479–502 (cit. on p. 19).

Maes, J., J. O. McClain, and L. N. Van Wassenhove (1991): Multilevel capacitated lot-sizing complexity and LP-based heuristics. *European Journal of Operational Research* 53.2, pp. 131–148 (cit. on p. 190).

Magnanti, T. L. and R. T. Wong (1984): Network design and transportation planning: Models and algorithms. *Transportation Science* 18.1, pp. 1–55 (cit. on p. 83).

Mansfield, R. S. (1996): Building competency models: Approaches for HR professionals. *Human Resource Management* 35.1, pp. 7–18 (cit. on pp. 19, 41).

Martello, S. and P. Toth (1990): *Knapsack problems: Algorithms and computer implementations*. Chichester: Wiley (cit. on p. 77).

Martí, R., J. M. Moreno-Vega, and A. Duarte (2010): Advanced multi-start methods. In: *Handbook of metaheuristics*. Ed. by M. Gendreau and J.-Y. Potvin. New York: Springer, pp. 265–281 (cit. on pp. 127, 157, 194).

Martí, R., M. G. C. Resende, and C. C. Ribeiro (2013): Multi-start methods for combinatorial optimization. *European Journal of Operational Research* 226.1, pp. 1–8 (cit. on p. 126).

Martino, J. P. (1995): *Research and development project selection*. New York: Wiley (cit. on pp. 87, 88).

McGrath, J. E. (1991): Time, interaction, and performance (TIP): A theory of groups. *Small Group Research* 22.2, pp. 147–174 (cit. on p. 29).

Miller, J. L. and L. S. Franz (1996): A binary-rounding heuristic for multi-period variable-task-duration assignment problems. *Computers & Operations Research* 23.8, pp. 819–828 (cit. on pp. 15, 190).

Mintzberg, H. (1980): Structure in 5's: A synthesis of the research on organization design. *Management Science* 26.3, pp. 322–341 (cit. on p. 28).

Muralidhar, K., R. Santhanam, and R. L. Wilson (1990): Using the analytic hierarchy process for information system project selection. *Information & Management* 18.2, pp. 87–95 (cit. on pp. 10, 88).

Nembhard, D. A. and M. V. Uzumeri (2000): An individual-based description of learning within an organization. *IEEE Transactions on Engineering Management* 47.3, pp. 370–378 (cit. on p. 66).

Nemhauser, G. L. and L. A. Wolsey (1999): *Integer and combinatorial optimization*. New York: Wiley (cit. on pp. 74, 76, 110, 112, 113).

Neumann, K. and M. Morlock (2002): *Operations Research*. 2nd ed. Munich: Hanser (cit. on pp. 54, 110, 121, 124, 145–147).

Neumann, K., C. Schwindt, and J. Zimmermann (2003): *Project scheduling with time windows and scarce resources*. 2nd ed. Berlin: Springer (cit. on p. 11).

Obreque, C., M. Donoso, G. Gutiérrez, and V. Marianov (2010): A branch and cut algorithm for the hierarchical network design problem. *European Journal of Operational Research* 200.1, pp. 28–35 (cit. on p. 110).

Oelsnitz, D. v. d. and M. W. Busch (2006): Social Loafing – Leistungsminderung in Teams. *Personalführung* 39.9, pp. 64–75 (cit. on p. 32).

Oesch, D. and J. Rodríguez Menés (2011): Upgrading or polarization? Occupational change in Britain, Germany, Spain and Switzerland, 1990–2008. *Socio-Economic Review* 9.3, pp. 503–531 (cit. on pp. 1, 18).

Orlin, J. B. (2013): Private communication via email, Apr. 15, 2013 (cit. on p. 183).

Otero, L. D., G. Centeno, A. J. Ruiz-Torres, and C. E. Otero (2009): A systematic approach for resource allocation in software projects. *Computers & Industrial Engineering* 56.4, pp. 1333–1339 (cit. on pp. 15–17).

Padberg, M. W., T. J. Van Roy, and L. A. Wolsey (1985): Valid linear inequalities for fixed charge problems. *Operations Research* 33.4, pp. 842–861 (cit. on p. 113).

Pan, P.-Q. (2008): Efficient nested pricing in the simplex algorithm. *Operations Research Letters* 36.3, pp. 309–313 (cit. on p. 255).

Patanakul, P., D. Z. Milosevic, and T. R. Anderson (2007): A decision support model for project manager assignments. *IEEE Transactions on Engineering Management* 54.3, pp. 548–564 (cit. on pp. 15, 16, 66, 98).

Pawlowsky, P., D. Menzel, and U. Wilkens (2005): Wissens- und Kompetenzerfassung in Organisationen. In: *Kompetenzmessung im Unternehmen*. Ed. by Arbeitsgemeinschaft Betriebliche Weiterbildungsforschung e. V. Münster: Waxmann, pp. 341–451 (cit. on p. 20).

Pinto, J. K. (2010): *Project management: Achieving competitive advantage*. 2nd ed. Upper Saddle River: Pearson (cit. on pp. 8, 9).

Reddy, W. B. and A. Byrnes (1972): Effects of interpersonal group composition on the problem-solving behavior of middle managers. *Journal of Applied Psychology* 56.6, pp. 516–517 (cit. on p. 67).

Reeves, G. R. and E. P. Hickman (1992): Assigning MBA students to field study project teams: A multicriteria approach. *Interfaces* 22.5, pp. 52–58 (cit. on pp. 15, 16).

Rieck, J., J. Zimmermann, and T. Gather (2012): Mixed-integer linear programming for resource leveling problems. *European Journal of Operational Research* 221.1, pp. 27–37 (cit. on pp. 69, 104).

Rüdrich, G. (2011): Projekte der Produktinnovation bewerten und selektieren. In: *Projektportfolio-Management*. Ed. by M. Hirzel, W. Alter, and M. Sedlmayer. Wiesbaden: Gabler, pp. 99–106 (cit. on p. 8).

Saaty, T. L. (1994): How to make a decision: The analytic hierarchy process. *Interfaces* 24.6, pp. 19–43 (cit. on p. 87).

Sahling, F. (2010): *Mehrstufige Losgrößenplanung bei Kapazitätsrestriktionen*. Wiesbaden: Gabler (cit. on p. 195).

Salewski, F. (1999): Heuristiken zur Dienstplanung bei flexibler Personalkapazität. *OR Spektrum* 21.3, pp. 361–379 (cit. on p. 283).

Salewski, F., A. Schirmer, and A. Drexl (1997): Project scheduling under resource and mode identity constraints: Model, complexity, methods, and application. *European Journal of Operational Research* 102.1, pp. 88–110 (cit. on p. 198).

Santhanam, R. and G. J. Kyparisis (1996): A decision model for interdependent information system project selection. *European Journal of Operational Research* 89.2, pp. 380–399 (cit. on pp. 10, 45).

Santhanam, R., K. Muralidhar, and M. Schniederjans (1989): A zero-one goal programming approach for information system project selection. *Omega* 17.6, pp. 583–593 (cit. on pp. 10, 88).

Santos, C., T. Gonzalez, H. Li, K.-Y. Chen, D. Beyer, S. Biligi, Q. Feng, R. Kumar, S. Jain, R. Ramanujam, and A. Zhang (2013): HP Enterprise Services uses optimization for resource planning. *Interfaces* 43.2 (cit. on pp. 15, 16).

Schaper, N. (2007): Arbeitsproben und situative Fragen zur Messung arbeitsplatzbezogener Kompetenzen. In: *Handbuch Kompetenzmessung*. Ed. by J. Erpenbeck and L. v. Rosenstiel. Stuttgart: Schäffer-Poeschel, pp. 160–174 (cit. on p. 20).

Schirmer, A. (1995): *A guide to complexity theory in operations research*. Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 381 (cit. on pp. 74, 76, 79).

Schirmer, A. and A. Drexl (2001): Allocation of partially renewable resources: Concept, capabilities, and applications. *Networks* 37.1, pp. 21–34 (cit. on p. 283).

Schlereth, T. (2009): *Project resource management in large organizations: A challenge for project management software*. Presentation at the meeting of the GOR working group "Projektmanagement und Scheduling", Projects in complex fast changing organisations: Selection, priorization, planning & controlling, Nov. 13, 2009, Munich (cit. on p. 40).

Schmidt, R. L. (1993): A model for R&D project selection with combined benefit, outcome and resource interactions. *IEEE Transactions on Engineering Management* 40.4, pp. 403–410 (cit. on pp. 10, 45).

Schneeweiß, C. (1992): Arbeitszeitmanagement und hierarchische Produktionsplanung. In: *Kapazitätsorientiertes Arbeitszeitmanagement*. Ed. by C. Schneeweiß. Heidelberg: Physica, pp. 7–22 (cit. on p. 54).

Schneider, B. and A. M. Konz (1989): Strategic job analysis. *Human Resource Management* 28.1, pp. 51–63 (cit. on p. 19).

Schneider, C. (2008): *Ressourcenmanagement in Matrixorganisationen und Multiprojektlandschaften*. Presentation at the meeting of the GOR working group „Projektmanagement und Scheduling", Apr. 4, 2008, Munich (cit. on p. 14).

Schneider, M., J. Grahl, D. Francas, and D. Vigo (2013): A problem-adjusted genetic algorithm for flexibility design. *International Journal of Production Economics* 141.1, pp. 56–65 (cit. on p. 25).

Schulte-Zurhausen, M. (1999): *Organisation*. 2nd ed. Munich: Vahlen (cit. on p. 40).

Schwarze, J. (2001): *Projektmanagement mit Netzplantechnik*. 8th ed. Herne: Neue Wirtschafts-Briefe (cit. on p. 42).

Sebok, A. (2000): Team performance in process control: Influences of interface design and staffing levels. *Ergonomics* 43.8, pp. 1210–1236 (cit. on p. 36).

Shanker, K. and Y.-J. J. Tzen (1985): A loading and dispatching problem in a random flexible manufacturing system. *International Journal of Production Research* 23.3, pp. 579–595 (cit. on p. 68).

Shapiro, A., D. Dencheva, and A. Ruszczyński (2009): *Lectures on stochastic programming*. Philadelphia: SIAM (Society for Industrial and Applied Mathematics) (cit. on p. 25).

Shippmann, J. S., R. A. Ash, M. Battista, L. Carr, L. D. Eyde, B. Hesketh, J. Kehoe, K. Pearlman, E. P. Prien, and J. I. Sanchez (2000): The practice of competency modeling. *Personnel Psychology* 53.3, pp. 703–740 (cit. on p. 19).

Simmel, G. (1902): The number of members as determining the sociological form of the group. II. *American Journal of Sociology* 8.2, pp. 158–196 (cit. on p. 30).

Slomp, J. and E. Molleman (2002): Cross-training policies and team performance. *International Journal of Production Research* 40.5, pp. 1193–1219 (cit. on pp. 15, 20, 21, 105, 106).

Souder, W. E. and T. Mandakovic (1986): R&D project selection models. *Research Management* 29.4, pp. 36–42 (cit. on p. 88).

Spitz-Oener, A. (2006): Technical change, job tasks, and rising educational demands: Looking outside the wage structure. *Journal of Labor Economics* 24.2, pp. 235–270 (cit. on p. 18).

Statista (2013): *Project and portfolio management software revenue 2008-2012, by vendor (fee-based)*. Statista GmbH, Hamburg. URL: `http://www.statista.com/statistics/268000/global-revenue-with-project-management-software-by-provider/` (visited on Apr. 9, 2014) (cit. on p. 8).

Süer, G. A. and R. R. Tummaluri (2008): Multi-period operator assignment considering skills, learning and forgetting in labour-intensive cells. *International Journal of Production Research* 46.2, pp. 469–493 (cit. on pp. 66, 283).

Sürie, C. and H. Stadtler (2003): The capacitated lot-sizing problem with linked lot sizes. *Management Science* 49.8, pp. 1039–1054 (cit. on p. 195).

Tavana, M. (2003): CROSS: A multicriteria group-decision-making model for evaluating and prioritizing advanced-technology projects at NASA. *Interfaces* 33.3, pp. 40–56 (cit. on pp. 10, 88).

Taylor, B. W., L. J. Moore, and E. R. Clayton (1982): R&D project selection and manpower allocation with integer nonlinear goal programming. *Management Science* 28.10, pp. 1149–1158 (cit. on pp. 9, 10, 88, 283).

Tett, R. P. and P. J. Murphy (2002): Personality and situations in co-worker preference: Similarity and complementarity in worker compatibility. *Journal of Business and Psychology* 17.2, pp. 223–243 (cit. on p. 67).

Thomas, E. J. and C. F. Fink (1963): Effects of group size. *Psychological Bulletin* 60.4, pp. 371–384 (cit. on p. 30).

Tiwari, V., J. H. Patterson, and V. A. Mabert (2009): Scheduling projects with heterogeneous resources to meet time and quality objectives. *European Journal of Operational Research* 193.3, pp. 780–790 (cit. on p. 12).

Tohidi, H. and M. J. Tarokh (2006): Productivity outcomes of teamwork as an effect of information technology and team size. *International Journal of Production Economics* 103.2, pp. 610–615 (cit. on p. 34).

Tuckman, B. W. and M. A. C. Jensen (1977): Stages of small-group development revisited. *Group & Organization Studies* 2.4, pp. 419–427 (cit. on p. 29).

Tukel, O. I. and W. O. Rom (1998): Analysis of the characteristics of projects in diverse industries. *Journal of Operations Management* 16.1, pp. 43–61 (cit. on p. 11).

Vairaktarakis, G. L. (2003): The value of resource flexibility in the resource-constrained job assignment problem. *Management Science* 49.6, pp. 718–732 (cit. on pp. 13, 24, 276).

Valls, V., A. Pérez, and S. Quintanilla (1996): A graph colouring model for assigning a heterogenous workforce to a given schedule. *European Journal of Operational Research* 90.2, pp. 285–302 (cit. on pp. 15, 16, 94, 95).

Valls, V., Á. Pérez, and S. Quintanilla (2009): Skilled workforce scheduling in service centres. *European Journal of Operational Research* 193.3, pp. 791–804 (cit. on pp. 12, 13, 49, 104, 105, 283).

Wallace, C. (2010): ZI round, a MIP rounding heuristic. *Journal of Heuristics* 16.5, pp. 715–722 (cit. on p. 190).

Walter, M. and J. Zimmermann (2010): A heuristic approach to project staffing. *Electronic Notes in Discrete Mathematics* 36 (Proceedings of the International Symposium on Combinatorial Optimization, Hammamet), pp. 775–782 (cit. on p. 187).

Walter, M. and J. Zimmermann (2012): On a multi-project staffing problem with heterogeneously skilled workers. In: *Operations Research Proceedings 2011*. Ed. by D. Klatte, H.-J. Lüthi, and K. Schmedders. Berlin: Springer, pp. 489–494 (cit. on pp. 216, 243).

Wayne, K. D. (2002): A polynomial combinatorial algorithm for generalized minimum cost flow. *Mathematics of Operations Research* 27.3, pp. 445–459 (cit. on p. 186).

Wegener, I. (2003): *Komplexitätstheorie: Grenzen der Effizienz von Algorithmen*. Berlin: Springer (cit. on pp. 74, 76, 109).

Weingartner, H. M. (1966): Capital budgeting of interrelated projects: Survey and synthesis. *Management Science* 12.7, pp. 485–516 (cit. on pp. 2, 10, 45).

Whitaker, R. A. (1985): Some add-drop and drop-add interchange heuristics for non-linear warehouse location. *Journal of the Operational Research Society* 36.1, pp. 61–70 (cit. on p. 149).

Wilke, H. A. M. and R. W. Meertens (1994): *Group performance*. London: Routledge (cit. on pp. 30, 33).

Williams, H. P. (1999): *Model building in mathematical programming*. 4th ed. Chichester: Wiley (cit. on pp. 59, 60, 69).

Winston, W. L. and J. B. Goldberg (2004): *Operations research: Applications and algorithms*. 4th ed. Belmont: Thomson Learning (cit. on pp. 110, 121).

Wolsey, L. A. (1998): *Integer programming*. New York: Wiley (cit. on pp. 110, 113, 189).

Wright, T. P. (1936): Factors affecting the cost of airplanes. *Journal of the Aeronautical Sciences* 3.4, pp. 122–128 (cit. on p. 65).

Wu, M.-C. and S.-H. Sun (2006): A project scheduling and staff assignment model considering learning effect. *International Journal of Advanced Manufacturing Technology* 28.11, pp. 1190–1195 (cit. on pp. 12–14, 65).

Yelle, L. E. (1979): The learning curve: Historical review and comprehensive survey. *Decision Sciences* 10.2, pp. 302–328 (cit. on p. 65).

Yoshimura, M., Y. Fujimi, K. Izui, and S. Nishiwaki (2006): Decision-making support system for human resource allocation in product development projects. *International Journal of Production Research* 44.5, pp. 831–848 (cit. on pp. 2, 10, 11, 15, 16, 41, 66, 88–90, 92, 283).

Zenger, T. R. and B. S. Lawrence (1989): Organizational demography: The differential effects of age and tenure distributions on technical communication. *Academy of Management Journal* 32.2, pp. 353–376 (cit. on pp. 29, 30, 33).

Zhang, S., L. Xie, and M. Adams (2005): An efficient data association approach to simultaneous localization and map building. *International Journal of Robotics Research* 24.1, pp. 49–60 (cit. on p. 190).

Zimmermann, J., C. Stark, and J. Rieck (2010): *Projektplanung: Modelle, Methoden, Management*. 2nd ed. Berlin: Springer (cit. on p. 42).