A Heuristic for the Stacker Crane Problem on Trees Which Is Almost Surely Exact

Amin Coja-Oghlan[1⋆], Sven O. Krumke[2⋆⋆], and Till Nierhoff[1⋆⋆⋆]

[1] Humboldt-Universität zu Berlin, Institut für Informatik
Unter den Linden 6, 10099 Berlin, Germany
{coja, nierhoff}@informatik.hu-berlin.de
[2] Konrad-Zuse-Zentrum für Informationstechnik Berlin, Department Optimization
Takustr. 7, 14195 Berlin-Dahlem, Germany
krumke@zib.de

Abstract. Given an edge-weighted transportation network G and a list of transportation requests L, the *stacker crane problem* is to find a minimum-cost tour for a server along the edges of G that serves all requests. The server has capacity one, and starts and stops at the same vertex. In this paper, we consider the case that the transportation network G is a tree, and that the requests are chosen randomly according to a certain class of probability distributions. We show that a polynomial time algorithm by Frederickson and Guan [11], which guarantees a 4/3-approximation in the worst case, on almost all inputs finds a minimum-cost tour, along with a certificate of the optimality of its output.

1 Introduction

The Stacker Crane Problem (SCP) is among the classical tour problems that have been studied since the early days of computing [12,17,3]. An illustrating application is scheduling a delivery truck [17]: Given transportation jobs that consist of a pickup and a delivery location, the truck must traverse a certain distance to complete the jobs. This distance depends on the order the truck chooses to serve the jobs, and the goal is to find an order minimizing the distance. In the following precise definition of the SCP, the order of the jobs is not explicit. However, it can be extracted from any Euler tour of $(V, L \cup A)$:

Definition 1 (Stacker Crane Problem SCP). *An instance of the Stacker Crane Problem* SCP *consists of an undirected graph $G = (V, E)$ with edge-lengths $\ell\colon E \to \mathbb{R}_0^+$ and a list L of pairs of vertices, called* requests. *A solution is a multi-set A of pairs (u, v) where $\{u, v\} \in E$ such that the directed multi-graph $(V, L \cup A)$ is Eulerian after removal of isolated vertices. The* cost *of A is the total length of an Euler tour in $(V, L \cup A)$, where the length of an arc (u, v) equals the length of a shortest path between u and v in G with respect to ℓ. We denote the cost of an optimum solution by* $\mathrm{SCP}(G, L)$.

⋆ Research supported by the German Science Foundation (DFG, grant FOR 413/1-1)
⋆⋆ Research supported by the German Science Foundation (DFG, grant Gr 883/10)
⋆⋆⋆ Research supported by the German Science Foundation (DFG, grant PR 296/6-3)

T. Ibaraki, N. Katoh, and H. Ono (Eds.): ISAAC 2003, LNCS 2906, pp. 605–614, 2003.

As it is a generalization of the TSP, it is clear that a polynomial time algorithm for the SCP is not likely to exist. There is an important special case of the SCP, where the underlying graph G is a tree. This case arises e.g. in applications to warehouses that are operated by automated item transportation devices (*stacker cranes*) [9,7]. The worst case tractability of the SCP on trees has been investigated thoroughly in [11]. In that paper it is shown that the SCP on trees is hard, and that there is a 5/4-approximation algorithm for it.

However, approximation is only one way to cope with NP-hardness. Another approach is to look for *heuristics* that solve the problem exactly on "all but a few" instances. In fact, one of the reasons to study approximation algorithms is that an algorithm that has a good performance guarantee in the worst case is expected to perform even better on a large number of instances [21, p. IX].

Average case analysis. The approach to solve hard computational problems on all but a few instances is also known as *average case* or *probabilistic analysis*. This approach has been applied successfully, for instance, to the minimum bisection problem [6], to the k-center-problem [19], and to the knapsack problem [5]. Frequently the investigation is motivated by the observation that a certain problem, despite being hard in the worst case, has an algorithm that solves virtually every instance arising from applications.

A more advanced issue is certifiability. Though an algorithm may compute a good or even optimal solution on almost all inputs, this does not necessarily mean that the algorithm also finds an efficiently checkable *certificate* that the solution is indeed reasonable. For instance, it is well-known that a linear-time greedy heuristic approximates the chromatic number of almost all graphs within a factor of 2 (cf. [16]). More precisely, by constructing a coloring of the input graph, the greedy heuristic computes an upper bound that is at most twice the chromatic number almost surely. However, as the greedy procedure does not provide a lower bound on the chromatic number, it fails to certify that the output is a good approximate solution. In contrast, the algorithms given in [6, 19,5] for minimum bisection, knapsack, and k-center provide certificates.

A crucial question in the context of average case analysis concerns the random model being used. To be reasonable and to explain real-world observations, the frequently changing input parameters can be modeled randomly, whereas static parameters, i.e. input parameters that vary not at all or only slightly, should be modeled in a worst case fashion, viz. deterministically. For instance, in the above application of the SCP to delivery trucks, the underlying graph models road distances which can be assumed to remain fixed values (for a longer time). Thus, it is not appropriate to model the network itself with a random experiment. On the other hand, the structure of the requests is different. In particular, if the truck has to be rescheduled often (say on a daily basis for the newly arrived demand), then it is reasonable to model the list of requests randomly.

Results. The aim of this paper is to investigate the average case of the SCP in the case where the underlying graph G is a tree. We consider the algorithm

"Use-minspan-tree" by Frederickson and Guan [11, Section 4.2]. We show that this algorithm is optimal on "all but a few" instances, thereby complementing the result of [11] that the algorithm is 4/3-approximative in the worst case. More precisely, our random model of the input assumes that the underlying tree is given deterministically (i.e. worst case-like), and that the list of requests is chosen at random.

Definition 2. *Let $T = (V, E)$ be a tree with arbitrary edge lengths $\ell : E \to \mathbb{R}_0^+$. Moreover, let $(p_v)_{v \in V}$ be an arbitrary probability distribution on the vertices of T, i.e. $p_v \in [0, 1]$ for all $v \in V$ and $\sum_{v \in V} p_v = 1$. Then a* random list L of requests *of length n is obtained by choosing each of the n requests at random as follows.*

- *The probability of the request $(u, v) \in V^2$ is $p_u \cdot p_v$.*
- *The requests are chosen independently of each other.*

Note that this model entails a wide variety of distributions, including the important special case of the uniform distribution (set $p_v = \#V^{-1}$ for all v). We say that an event A holds *with high probability* or *whp.* if the probability that A occurs tends to 1 as the number n of requests tends to infinity. Thus, the notion "with high probability" formalizes the naïve phrase "on all but a few instances". Note that the relation to the size of the underlying network is *not* relevant. The aim of this paper is to prove the following theorem.

Theorem 3. *The algorithm "Use-minspan-tree" by Frederickson and Guan (Algorithm 9 below) solves instances that are chosen according to the random model given in Definition 2 optimally with high probability. This is certifiable, i.e. whp. a random instance has a certain property which (i) can be checked efficiently and (ii) implies the optimality of the algorithm's solution.*

A detailed description of the algorithm is the content of Section 2. The proof of Theorem 3 starts with some preliminaries in Section 2, and its sketch will be completed in Section 4. In Section 3 we give an alternative description of the random model of Definition 2, which is essential for the proof of Theorem 3.

Related work. Approximation algorithms for the SCP on general graphs can achieve an approximation ratio of 129/128 at the best, if P $\neq$ NP [20]. The best approximation ratio known for the case of general underlying graphs is 9/5 and is achieved by an algorithm of Frederickson, Hecht, and Kim [12]. The complete algorithm by Frederickson and Guan [11] achieves an approximation ratio of 5/4 on trees. On general trees the problem is NP-hard [11,9], but on paths, the problem is in P [2].

Several extensions of the SCP are known as *dial-a-ride problems* or DARP. Of these, the SCP is the special case of a single server with capacity one. Charikar and Rhaghavachari [8] consider the case of the capacitated DARP, i.e., where the server can serve a limited number of jobs simultaneously. This problem is already NP-hard on paths if no preemption is allowed, but solvable in polynomial time

in the preemptive case [14]. The paper [15] considers the DARP when additional precedence constraints between the requests are specified. Online variants of the DARP where requests become known to an online algorithm over time have been studied in [10,1,4]. The online scenario, as well as release time constraints are beyond the scope of this paper.

The average case of the SCP on trees has been examined for the special case of caterpillars as underlying graphs in [9]. (This special case is important for the application to the scheduling of elevators with starting and stopping times.) The elevator scheduling problem with capacities but without starting and stopping times has been solved by Karp (see the discussion by Knuth [18, Section 5.4.8]).

2 The Algorithm

In the following we give a rough sketch of the algorithm "Use-minspan-tree" of Frederickson and Guan that is quoted in Theorem 3. It has two main parts that we call *balancing* and *MST*. Balancing adds new requests to the given list of requests in such a way that the optimum cost does not increase. Therefore, an optimum solution with respect to the extended list of requests is also an optimum solution to the original instance.

Algorithm 4 (Balancing).
Input: A tree $T = (V, E)$ and a list of requests $L \subset (V \times V)^*$.
Output: A multiset of additional requests $B \subset (V \times V)^*$.
Each edge $e = \{u, v\} \in E$ corresponds to a cut $(C, V \setminus C)$, where $C \ni u$. Let e_+ be the number of requests starting in C and ending in $V \setminus C$, i.e. in $C \times V \setminus C$. Let e_- be the number of requests in $V \setminus C \times C$. Assume that $e_+ \geq e_-$. For e, B contains $e_+ - e_-$ requests (u, v). B can be computed in linear time [11].

Proposition 5 ([11]). *Let T be a tree, L be a given list of requests, and B be the output of the balancing procedure. Then* $\mathrm{SCP}(T, L) = \mathrm{SCP}(T, L \cup B)$ *and every component of $(V, L \cup B)$ is Eulerian.* □

We call the isolated vertices of $(V, L \cup B)$ the *trivial components* and all other components the *nontrivial components*. Note that the remaining task is to find a minimum cost (multi-)set A of pairs so that $(V, L \cup B \cup A)$ has only one nontrivial, Eulerian component. It is easy to see that such a multiset consists of anti-parallel pairs along a subset of those edges of the tree that are not internal to one of the components of $(V, L \cup B)$.

Definition 6 (star metric). *Let $\mathcal{C}$ be the set of components of $(V, L \cup B)$ (including isolated vertices). For $C_1, C_2 \in \mathcal{C}$ let their distance, $d(C_1, C_2)$, be the length of a shortest path connecting a vertex of C_1 to a vertex of C_2. Then $(\mathcal{C}, d)$ is a metric and we denote the set of nontrivial components by $\mathcal{C}'$. Using somewhat unconventional notation, we call $(\mathcal{C}, d)$ a* star metric, *if there is a nontrivial component $C^* \in \mathcal{C}'$, such that $d(C_1, C_2) = d(C_1, C^*) + d(C^*, C_2)$ for all $C_1, C_2 \in \mathcal{C}'$.*

The optimum solution A has anti-parallel arcs along a minimum cost tree in $(\mathcal{C}, d)$ that spans all nontrivial components. The problem to find such a tree is known as the Steiner tree problem, and the second part of the algorithm is essentially the well-known *minimum spanning tree heuristic* for that problem.

Algorithm 7 (MST).
Input: The metric $(\mathcal{C}, d)$ with a subset $\mathcal{C}' \subset \mathcal{C}$ of nontrivial components
Output: A set A of anti-parallel arcs along the edges of a tree in $(\mathcal{C}, d)$ that spans $\mathcal{C}'$ and results from the minimum spanning tree heuristic for the Steiner tree problem.

Proposition 8. *The set A output by Algorithm 7 is at most twice as long as a minimum cost set. It is of minimum cost, if the metric $(\mathcal{C}, d)$ is a star metric.* □

This proposition is a consequence of elementary results on approximation algorithms for the Steiner tree problem in graphs, see e.g. [13]. Then the algorithm of Frederickson and Guan that is quoted in Theorem 3 reads as follows:

Algorithm 9 (Use-minspan-tree [11, Section 4.2]).
Input: A tree T, a list L
Output: The length of an Euler tour in $(V, L \cup B \cup A)$, where
B is the output of Algorithm 4 on input T and L, and
A is the output of Algorithm 7 on the input induced by $(V, L \cup B)$.

As a direct consequence of Proposition 8, this algorithm has a performance ratio of 2 and is optimal (or exact) if the metric $(\mathcal{C}, d)$ is a star metric. This favorable situation can be detected easily in polynomial time. To complete the proof of Theorem 3, we show that it occurs with high probability. Our key to proving Theorem 3 is the following result:

Theorem 10. *Whp. $(\mathcal{C}, d)$ is a star metric.*

The following two sections are devoted to the proof of Theorem 10. First we show that it suffices to consider the case that T is a full binary tree.

3 From General Trees to Binary Trees

In this section we show that we can simplify the situation of a general tree to binary trees. Specifically, given a list of requests L, we shall modify the tree T via some elementary operations so that we obtain a full binary tree with exactly one request starting or ending at each leaf, and without requests incident with internal vertices. Recall that a binary tree is *full*, if each non-leaf node has exactly two children. Indeed, we just apply the following four operations repeatedly until T is of the desired form.

a. Remove leaves that are not involved in requests.
b. Introduce a new vertex for each end of a request and append it as a leaf to the original endpoint of the request. The new edge gets $\ell = 0$.

c. Contract vertices of degree 2. If v is a vertex of degree 2 and e, f are the incident edges, then let $\ell(e) \leftarrow \ell(e) + \ell(f)$, $\ell(f) \leftarrow 0$, and identify v with the other endpoint of f.
d. Split vertices of degree $d > 3$ into $d - 2$ vertices of degree 3 each. If v is a vertex of degree $d > 3$ and e, f are two of the incident edges, then let v' be a new vertex, connect it to v with an edge of $\ell = 0$ and replace v in e and f with v'. Proceed until v has degree 3.

Having adapted the tree T via a–d, we place a *root* r in the middle of an edge both of whose endpoints are connected to at least a third of the leaves without using that edge. We call the outcome (T_L, L) of the above procedure the *modified instance*. We emphasize that passing to the modified instance is part of the proof of Theorem 10; the *algorithm* sticks to the original instance.

Lemma 11. *If after applying the balancing operation the non-trivial components of the modified instance (T_L, L) form a star metric, then so do the non-trivial components of the original instance (T, L).*

Let L be a list of requests of length n. For each vertex v of T we let $d_v(L)$ denote the number of occurrences of v in L. Moreover, let $d(L) = (d_v(L))_{v \in V}$. Clearly, the outcome (T_L, L) of the modification procedure a–d depends only on the sequence $d(L)$. Conversely, for each sequence $d = (d_v)_{v \in V}$ such that $\sum_v d_v = 2n$ and $d_v \geq 0$ for all v, we can construct a rooted binary tree T_d via a–d, which has precisely $2n$ leaves $l_1, \ldots, l_{2n}$. Further, fixing d, a random list L of requests conditioned on $d(L) = d$ induces a random set of arcs $R(L) \subset \{l_1, \ldots, l_{2n}\}^2$ in which each leaf occurs precisely once. One could call $R(L)$ a *perfect directed matching* of the leaves $l_1, \ldots, l_{2n}$.

However, fixing d (and thus T_d), we can generate a random perfect directed matching R without referring to transportation requests in the original tree T at all: First, we choose a sequence $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{2n}) \in \{1, -1\}^{2n}$ such that $\sum_{i=1}^{2n} \boldsymbol{x}_i = 0$ uniformly at random among all $\binom{2n}{n}$ possible sequences. The sequence $\boldsymbol{x}$ specifies which leaves l_j will be heads ($x_j = 1$) and which leaves l_i will be tails ($x_i = -1$) of arcs in R. Then, we choose a permutation $\boldsymbol{\sigma} : \{1, \ldots, n\} \to \{1, \ldots, n\}$ uniformly at random and independently of $\boldsymbol{x}$. As there are precisely n leaves l_j with $x_j = 1$ and precisely n leaves l_i with $x_i = -1$, the permutation $\boldsymbol{\sigma}$ induces a bijection $\boldsymbol{\sigma}_{\boldsymbol{x}} : \{l_i |\ x_i = -1\} \to \{l_j |\ x_j = 1\}$ in the obvious manner. Finally, the perfect directed matching $R(\boldsymbol{x}, \boldsymbol{\sigma})$ is just $\{(l_i, l_j) |\ \boldsymbol{\sigma}_{\boldsymbol{x}}(l_i) = l_j\}$. (Note the similarity between $R(\boldsymbol{x}, \boldsymbol{\sigma})$ and the configuration model of random graphs with a prescribed degree sequence [16].) A tedious counting argument proves the following lemma.

Lemma 12. *Fix d. The probability distribution on perfect directed matchings induced by $R(\boldsymbol{x}, \boldsymbol{\sigma})$ and by $R(L)$ conditioned on $d(L) = d$ coincide.*

By Lemma 11 and Lemma 12, in order to prove Theorem 10 it suffices to show the following.

Theorem 13. *Let T be any full binary tree with leaves $l_1, \ldots, l_{2n}$. Let $R = R(\boldsymbol{x}, \boldsymbol{\sigma})$. Let B be the output of the balancing operation on input (T, R). Then*

whp. the non-trivial components of the graph $D(\boldsymbol{x}, \boldsymbol{\sigma}) = (V, R \cup B)$ *form a star metric.*

4 Proof of Theorem 13

After introducing some notation and stating some simple lemmas in Section 4.1, we proceed in two steps to prove Theorem 13. First, in Section 4.2 we analyze the number and the size of the components of the graph $D(\boldsymbol{x}, \boldsymbol{\sigma})$. It turns out that though there may be a few "small" components, there is exactly one "large" component whp. Then, in Section 4.3, we show that in the tree T any path connecting two "small" components will pass through the "large" component whp., thereby establishing that the components form a star metric.

4.1 Preliminaries

We let $\log = \log_2$. Throughout, we let $T = (V, E)$ be a full binary tree with root r and leaves $l_1, \ldots, l_{2n}$. We have a canonical partial order $\preceq$ on V, by letting $v \preceq w$ iff w lies on the path from v to r. For each $v \in V \setminus \{r\}$, we let $\rho(v)$ denote the parent of v. We say that a set $S \subset V$ *majorizes* $S' \subset V$ ($S' \preceq S$) if for each $w \in S'$ there is $v \in S$ such that $w \preceq v$. For each $v \in V$, let $b(v)$ denote the number of leaves majorized by v.

As in Section 3, we let $\boldsymbol{x}$ denote a sequence $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{2n}) \in \{1, -1\}^{2n}$ such that $\sum_i \boldsymbol{x}_i = 0$ chosen uniformly at random. For each $v \in V$, let $S_v = S_v(\boldsymbol{x}) = \sum_{i: l_i \preceq v} \boldsymbol{x}_i$. We say that v is *balanced* if $S_v = 0$. Set $E_B = E_B(\boldsymbol{x}) = \{\{v, \rho(v)\} |\ v \in V \setminus \{r\} \text{ is not balanced}\}$, and $B = B(\boldsymbol{x}) = (V, E_B)$. Moreover, let $\boldsymbol{\sigma}$ denote a permutation $\{1, \ldots, n\} \to \{1, \ldots, n\}$ chosen uniformly at random and independently of $\boldsymbol{x}$, and let $\boldsymbol{\sigma}_{\boldsymbol{x}}$ be as in Section 3. Let $E_A = \{\{l_i, l_j\} |\ \boldsymbol{\sigma}_{\boldsymbol{x}}(l_i) = l_j\}$, and $A = A(\boldsymbol{x}, \boldsymbol{\sigma}) = (V, E_A \cup E_B)$. Note that E_B consists of precisely those edges of T that will occur in the output of the balancing operation on input $R(\boldsymbol{x}, \boldsymbol{\sigma})$. Therefore, the non-trivial components of the simple graph A are in one-to-one correspondence with the non-trivial components of the directed multigraph $D(\boldsymbol{x}, \boldsymbol{\sigma})$ (cf. Theorem 13). First, we compute the probability that $S_v = 0$.

Lemma 14. *For all* $v \neq r$ *we have* $\mathrm{P}(S_v = 0) = O(b(v)^{-1/2})$.

Let $\mathcal{C} = \mathcal{C}(\boldsymbol{x})$ denote the set of all non-trivial components of B.

Lemma 15. *Each* $C \in \mathcal{C}$ *contains at least two leaves and has a unique maximal vertex (w.r.t. the partial order* $\preceq$*), which is balanced.*

We say that a set $\mathcal{C}_0 \subset \mathcal{C}$ is *separated* in A if there is no edge in E_A that joins a vertex in $\mathcal{C}_0$ with a vertex outside $\mathcal{C}_0$. Let $b(\mathcal{C}_0)$ denote the number of leaves $l_j \in \mathcal{C}_0$ with $\boldsymbol{x}_j = 1$.

Lemma 16. *The probability that* $\mathcal{C}_0$ *is separated is* $\binom{n}{b(\mathcal{C}_0)}^{-1}$.

It will be useful to partition the set V into sets

$$L_j = \{v \in V |\ 2^j \leq b(v) < 2^{j+1}\} \qquad (j = 0, \ldots, \log(2n)).$$

Then the sets L_j are pairwise disjoint, and their union is V. L_0 is the set of all leaves of T and it may be helpful to think of the sets L_j as "levels" in the tree T. As a convenience we set

$$L_{\leq k} = L_1 \cup \cdots \cup L_k \text{ and } L_{\geq k} = L_k \cup \cdots \cup L_{\log(2n)} \qquad (1 \leq k \leq \log(2n)).$$

Lemma 17. *Let $1 \leq j \leq \log(2n)$. The set L_j contains at most $n2^{1-j}$ maximal (minimal) vertices (w.r.t. the partial order $\preceq$).*

4.2 The Effect of the Balancing Operation

Let us call a non-trivial component of A *small* if it contains at most n leaves of T. Otherwise, we call the non-trivial component *large*.

Proposition 18. *Whp. the graph A enjoys the following properties.*

1. *There is a unique large component, which contains $(2 - o(1))n$ leaves.*
2. *There are $O(\log(n))$ small components.*
3. *All small components are cycles of length $O(\log(n))$.*

The proof of Proposition 18 proceeds in two steps. First, we estimate the number of non-trivial components of the graph $B = B(\boldsymbol{x}) = (V, E_B)$, i.e. we only consider edges included via the balancing operation. Then, we study how the requests E_A connect the non-trivial components of B into larger blocks.

The number of non-trivial components of B. Since each non-trivial component of B contains a unique maximal vertex, we can define $X_j = X_j(\boldsymbol{x})$ to be the number of non-trivial components of B whose maximal vertex belongs to L_j. Then $\sum_{j\geq 1} X_j$ is the total number of non-trivial components of B.

How large can the probable value of $\sum_{j\geq 1} X_j$ be? If T is a complete binary tree, we see that the expectation of X_1 can be as large as $n/2$. Furthermore, in this case, X_1 is concentrated about its mean, whence $\sum_{j\geq 1} X_j \geq (1 - o(1))n/2$ whp. On the other extreme, in the case where T is a caterpillar, the analogy with a constrained random walk shows that $\sum_{j\geq 1} X_j = O(n^{1/2})$ whp.

As a consequence, we cannot expect that the number of non-trivial components is sublinear in n. Nonetheless, it can be shown, that the number is not more than a constant fraction of n:

Lemma 19. *The number of non-trivial components of B is at most $98n/99$ whp.*

Sketch of proof. First, using a first moment argument, one can show that only very few components reach the high levels of T. More precisely, $\sum_{j_0<j} X_j(\boldsymbol{x}) = o(n)$ whp., where $j_0 = \log(2n)/3$. Then a careful analysis shows, that the expectation of the number $\sum_{j=1}^{j_0} X_j$ of the other non-trivial components of B, is bounded by $49n/50$. Finally, a martingale argument proves that this number is concentrated about its mean. □

Glueing the non-trivial components of B together. In order to prove Proposition 18, we show that the edges E_A induced by the random permutation $\boldsymbol{\sigma}$ link most of the non-trivial components of B. We rely on the fact that $\boldsymbol{x}$ and $\boldsymbol{\sigma}$ are chosen independently.

Lemma 20. *The number of non-trivial components of A is $O(\log n)$ whp.*

Proof. By Lemma 19, we may assume that the number of non-trivial components of $B = B(\boldsymbol{x})$ is $\leq 98n/99$. Observe that the number of non-trivial components of A is at most twice the number of subsets $\mathcal{C}_0 \subset \mathcal{C}$ that are separated in A and that satisfy $b(\mathcal{C}_0) \leq n/2$. (Actually, the number of separated sets $\mathcal{C}_0$ is exponential in the number of non-trivial components of A, but we don't need that.) Therefore, invoking Lemma 16, we see that the expected number of non-trivial components of A is at most

$$2\sum_{k=1}^{\#\mathcal{C}-1} \sum_{\mathcal{C}_0\subset\mathcal{C},\ \#\mathcal{C}_0=k,\ b(\mathcal{C}_0)\leq n/2} \binom{n}{b(\mathcal{C}_0)}^{-1} \leq 2\sum_{k} \binom{98n/99}{k}\binom{n}{k}^{-1} = O(1).$$

Thus, by Markov's inequality, the number of non-trivial components of A is $O(\log n)$ with high probability. □

Proof of Proposition 18 (sketch). The above lemma already gives the second assertion in Proposition 18. The proof of the third claim uses a similar estimate. Finally, as there are at most $O(\log n)$ small components all of which are cycles whp., at most $O(\log n)^2$ leaves lie in small components. Hence, there must be a large component that contains the remaining $(2 - o(1))n$ leaves. □

4.3 Completing the Proof of Theorem 13

To prove that the non-trivial components of A form a star metric whp., we show that any path that joins two vertices in different small components of A passes through the large component.

Lemma 21. *There is no vertex $v \in L_{\leq 9\log(2n)/10}$ that majorizes vertices of two different small components of A whp.*

Lemma 22. *Let $j = 9\log(2n)/10$. The graph A enjoys the following property whp. If $u, w \in L_{\geq j}$ are such that $u \preceq w$, and if u majorizes vertices that belong to a small component C_1, and w majorizes vertices that belong to a small component $C_2 \neq C_1$ of A, then there is a vertex s, $u \preceq s \preceq w$, such that $|S_s| > 2$.*

Proof of Theorem 13. We may assume that A enjoys the properties stated in Proposition 18, Lemma 21, and Lemma 22. Let $C_1 \neq C_2$ be small components of A, and let C be the unique large component. We are to show that any path P that connects $v \in C_1$ with $w \in C_2$ passes through C. If $v \in L_{\leq 9\log(2n)/10}$, then by Lemma 21 the path P will pass through $L_{\geq 9\log(2n)/10}$. Thus, let s be the first vertex in $L_{\geq 9\log(2n)/10}$ on P, let q be the last one, and let u be the maximal vertex on P. Then $v \preceq s$, $w \preceq q$, and $q, s \preceq u$. Therefore, by Lemma 22, there exists a vertex t on P such that $|S_t| > 2$. Since by Proposition 18 all small components are cycles, we conclude that $t \in C$. □

References

1. Ascheuer, N., Krumke, S.O., Rambau, J.: Online dial-a-ride problems: Minimizing the completion time. In: Proceedings of the 17th STACS. Volume 1770 of Lecture Notes in Computer Science., Springer (2000) 639–650
2. Atallah, M.J., Kosaraju, S.R.: Efficient solutions to some transportation problems with applications to minimizing robot arm travel. SIAM Journal on Computing **17** (1988) 849–869
3. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and Approximation. Springer (1999)
4. Ausiello, G., Feuerstein, E., Leonardi, S., Stougie, L., Talamo, M.: Algorithms for the on-line traveling salesman. Algorithmica **29** (2001) 560–581
5. Beier, R., Vöcking, B.: Random knapsack in expected polynomial time. In: Proc 35th SToC. (2003)
6. Boppana, R.B.: Eigenvalues and graph bisection: An average case analysis. In: Proceedings of 28th FoCS. (1987)
7. Burkard, R., Fruhwirth, B., Rote, G.: Vehicle routing in an automated warehouse: Analysis and optimization. Annals of Operations Research **57** (1995) 29–44
8. Charikar, M., Raghavachari, B.: The finite capacity dial-A-ride problem. In: Proceedings of the 39th FoCS. (1998)
9. Coja-Oglan, A., Krumke, S.O., Nierhoff, T.: Scheduling a server on a caterpillar network - a probabilistic analysis. In: Proceedings of the 6th Workshop on Models and Algorithms for Planning and Scheduling Problems (2003)
10. Feuerstein, E., Stougie, L.: On-line single server dial-a-ride problems. Theoretical Computer Science. To appear.
11. Frederickson, G.N., Guan, D.J.: Nonpreemptive ensemble motion planning on a tree. Journal of Algorithms **15** (1993) 29–60
12. Frederickson, G.N., Hecht, M.S., Kim, C.E.: Approximation algorithms for some routing problems. SIAM Journal on Computing **7** (1978) 178–193
13. Gröpl, C., Hougardy, S., Nierhoff, T., Prömel, H.J.: Approximation algorithms for the Steiner tree problem in graphs. In Cheng, X., Du, D.Z., eds.: Steiner Trees in Industry. Kluwer Academic Publishers (2001) 235–279
14. Guan, D.J.: Routing a vehicle of capacity greater than one. Discrete Applied Mathematics **81** (1998) 41–57
15. Hauptmeier, D., Krumke, S.O., Rambau, J., Wirth, H.C.: Euler is standing in line. Discrete Applied Mathematics **113** (2001) 87–107
16. Janson, S., Luczak, T., Ruciński, A.: Random Graphs. John Wiley & Sons (2000)
17. Johnson, D.S., Papadimitriou, C.H.: Performance guarantees for heuristics. In Lawler, E.L., Lenstra, J.K., Kan, A.H.G.R., Shmoys, D.B., eds.: The Travelling Salesman Problem. Wiley (1985) 145–180
18. Knuth, D.E.: The art of computer programming. Volume 3: Sorting and searching. Addison-Wesley, Reading MA (1968)
19. Kreuter, B., Nierhoff, T.: Greedily approximating the r-independent set and k-center problems on random instances. In Rolim, J.D.P., ed.: Randomization and Approximation Techniques in Computer Science. Volume 1269 of LNCS., Springer (1997) 43–53
20. Papadimitriou, C.H., Vempala, S.: On the approximability of the traveling salesman problem. In: Proc. of the 32nd SToC. (2000)
21. Vazirani, V.: Approximation Algorithms. Springer (2001)