Deterministic Rendezvous in Graphs

Anders Dessmark[1], Pierre Fraigniaud[2], and Andrzej Pelc[3]

[1] Dept. of Computer Science, Lund Univ., Box 118, S-22100 Lund, Sweden. andersd@cs.lth.se
[2] CNRS, LRI, Univ. Paris Sud, 91405 Orsay, France. http://www.lri.fr/~pierre
[3] Dép. d'Informatique, Univ. du Québec en Outaouais, Hull, Québec J8X 3X7, Canada. pelc@uqo.ca

Abstract. Two mobile agents having distinct identifiers and located in nodes of an unknown anonymous connected graph, have to meet at some node of the graph. We present fast *deterministic* algorithms for this rendezvous problem.

1 Introduction

Two mobile agents located in nodes of a network, modeled as an undirected connected graph, have to meet at some node of the graph. This task is known as the rendezvous problem in graphs, and in this paper we seek efficient deterministic algorithms to solve it. If nodes of the graph are labeled then agents can decide to meet at a predetermined node and the rendezvous problem reduces to graph exploration. However, in many applications, when rendezvous is needed in an unknown environment, such unique labeling of nodes may not be available, or limited sensory capabilities of the agents may prevent them from perceiving such labels. Hence it is important to be able to program the agents to explore *anonymous* graphs, i.e., graphs without unique labeling of nodes. Clearly, the agents have to be able to *locally* distinguish ports at a node: otherwise, an agent may even be unable to visit all neighbors of a node of degree 3 (after visiting the second neighbor, the agent cannot distinguish the port leading to the first visited neighbor from that leading to the unvisited one). Consequently, agents initially located at two nodes of degree 3, might never be able to meet. Hence we make a natural assumption that all ports at a node are locally labeled $1, \ldots, d$, where d is the degree of the node. No coherence between those local labelings is assumed. We also do not assume any knowledge of the topology of the graph or of its size. Likewise, agents are unaware of the distance separating them.

Agents move in synchronous rounds. In every round, an agent may either remain in the same node or move to an adjacent node. We consider two scenarios: *simultaneous startup*, when both agents start executing the algorithm at the same time, and *arbitrary startup*, when starting times are arbitrarily decided by the adversary. In the former case, agents know that starting times are the same, while in the latter case, they are not aware of the difference between starting times, and each of them starts executing the rendezvous algorithm and counting rounds since its own startup. The agent who starts earlier and happens to visit the starting node of the later agent *before* the startup of this later agent, is not aware of this fact, i.e, we assume that agents are created at their startup time and not waiting in the node before it.

An agent, currently located at a node, does not know the other endpoints of yet unexplored incident edges. If the agent decides to traverse such a new edge, the choice of the actual edge belongs to the adversary, as we are interested in the worst-case performance. We assume that, if agents get to the same node in the same round, they become aware of it and rendezvous is achieved. However, if agents cross each other along an edge (moving in the same round along the same edge in opposite directions) they do not notice this fact. In particular, rendezvous is not possible in the middle of an edge. The *time* used by a rendezvous algorithm, for a given initial location of agents in a graph, is the worst-case number of rounds since the startup of the later agent until rendezvous is achieved, where the worst case is taken over all adversary decisions, whenever an agent decides to explore a new edge adjacent to a currently visited node, and over all possible startup times (decided by the adversary), in case of the arbitrary startup scenario.

If agents are identical, i.e., they do not have distinct identifiers, and execute the same algorithm, then deterministic rendezvous is impossible even in the simplest case when the graph consists of two nodes joined by an edge, agents are initially located at both ends of it and start simultaneously: in every round both agents will either stay in different nodes or will both move to different nodes, thus they will never meet. Hence we assume that agents have distinct identifiers, called labels, which are two different integers written as binary strings starting with 1, and that every agent knows its own label. Now, if *both* agents knew *both* labels, the problem can be again reduced to that of graph exploration: the agent with smaller label does not move, and the other agent searches the graph until it finds it. However, the assumption that agents know each other may often be unrealistic: agents may be created in different parts of the graph in a distributed fashion, oblivious of each other. Hence we assume that each agent knows its own label but does not know the label of the other. The only initial input of a (deterministic) rendezvous algorithm executed by an agent is the agent's label. During the execution of the algorithm, an agent learns the local port number by which it enters a node and the degree of the node.

In this setting, it is not even obvious that (deterministic) rendezvous is at all possible. Of course, if the graph has a distinguished node, e.g., a unique node of a given degree, agents could decide to meet at this node, and hence rendezvous would be reduced to exploration (note that an agent visiting a node becomes aware of its degree). However, a graph may not have such a node, or its existence may be unknown and hence impossible to use in the algorithm. For example, it does not seem obvious *apriori* if rendezvous can be achieved in a ring.

The following are the two main questions guiding our research:

Q1. Is rendezvous feasible in arbitrary graphs?

Q2. If so, can it be performed efficiently, i.e., in time polynomial in the number n of nodes, in the difference τ between startup times and in labels L_1, L_2 of the agents (or even polynomial in n, τ and $\log L_1$, $\log L_2$)?

Our results. We start by introducing the problem in the relatively simple case of rendezvous in trees. We show that rendezvous can be completed in time $O(n + \log l)$ on any n-node tree, where l is the smaller of the two labels, even with arbitrary startup. We also show that for some trees this complexity cannot be improved, even with simultaneous startup. Trees are, however, a special case from the point of view of the rendezvous problem, as any tree has either a central node or a central edge, which facilitates the meeting (incidentally, the possibility of the second case makes rendezvous not quite trivial, even in trees). As soon as the graph contains cycles, the technique which we use for trees cannot be applied. Hence it is natural to concentrate on the simplest class of such graphs, i.e., rings. We prove that, with simultaneous startup, optimal time of rendezvous on any ring is $\Theta(D \log l)$, where D is the initial distance between agents. We construct an algorithm achieving rendezvous with this complexity and show that, for any distance D, it cannot be improved. With arbitrary startup, $\Omega(n + D \log l)$ is a lower bound on the time required for rendezvous on an n-node ring. Under this scenario, we show two rendezvous algorithms for the ring: an algorithm working in time $O(n \log l)$, for known n, and an algorithm polynomial in n, l and the difference τ between startup times, if n is unknown. For arbitrary graphs, our main contribution is a general feasibility result: rendezvous can be accomplished on arbitrary connected graphs, even with arbitrary startup. If simultaneous startup is assumed, we construct a generic rendezvous algorithm, working for all connected graphs, which is optimal for the class of graphs of bounded degree, if the initial distance between agents is bounded.

Related work. The rendezvous problem has been introduced in [16]. The vast literature on rendezvous (see the book [3] for a complete discussion and more references) can be divided into two classes: papers considering the geometric scenario (rendezvous in the line, see, e.g., [10,11,13], or in the plane, see, e.g., [8, 9]), and those discussing rendezvous in graphs, e.g., [1,4]. Most of the papers, e.g., [1,2,6,10] consider the probabilistic scenario: inputs and/or rendezvous strategies are random. A natural extension of the rendezvous problem is that of gathering [12,15,17], when more than 2 agents have to meet in one location. To the best of our knowledge, the present paper is the first to consider deterministic rendezvous in unlabeled graphs assuming that each agent knows only its own identity.

Terminology and notation. Labels of agents are denoted by L_1 and L_2. The agent with label L_i is called agent i. (An agent does not know its number, only its label). Labels are distinct integers represented as binary strings starting with 1. l denotes the smaller of the two labels. The difference between startup times of the agents is denoted by τ. The agent with earlier startup is called the *earlier* agent and the other agent is called the *later* agent. In the case of simultaneous startup, the earlier agent is defined as agent 1. (An agent does not know if it is earlier or later). We use the word "graph" to mean a simple undirected connected graph. n denotes the number of nodes in the graph, Δ the maximum degree, and D the distance between initial positions of agents.

2 Rendezvous in Trees

We introduce the rendezvous problem in the relatively simple case of trees. In this section we assume that agents know that they are in a tree, although they know neither the topology of the tree nor its size. Trees have a convenient feature from the point of view of rendezvous. Every tree has either a *central node*, defined as the unique node minimizing the distance from the farthest leaf, or a *central edge*, defined as the edge joining the only two such nodes. This suggests an idea for a natural rendezvous algorithm, even for arbitrary startup: explore the tree, find the central node or the central edge, and try to meet there. Exploring the tree is not a problem: an agent can perform DFS, keeping a stack for used port numbers. At the end of the exploration, the agent has a map of the tree, can identify the central node or the central edge, and can find its way either to the central node or to one endpoint of the central edge, in the latter case knowing which port corresponds to the central edge. In the first case, rendezvous is accomplished after the later agent gets to the central node. In the second case, the rendezvous problem in trees can be reduced to rendezvous on graph K_2 consisting of two nodes joined by an edge. We now show a procedure for rendezvous in this simplest graph.

The Procedure Extend-Labels, presented below, performs a rendezvous on graph K_2 in the model with arbitrary startup. The procedure is formulated for an agent with label L. Enumerate the bits of the binary representation of L from left to right, i.e., starting with the most significant bit. The actions taken by agents are either *move* (i.e., traverse the edge) or *stay* (i.e., remain in place for one round). Rounds are counted from the starting round of the agent. Intuitively, the behavior of the agent is the following. First, transform label L into the string L^* by writing bits 10 and then writing twice every bit of L. Then repeat indefinitely string L^*, forming an infinite binary string. The agent moves (stays) in round i, if the ith position of this infinite string is 1 (0). Below we give a more formal description of the algorithm.

Procedure Extend-Labels
In round 1 *move.*
In round 2 *stay.*
In rounds $3 \le i \le 2\lfloor \log L \rfloor + 4$, *move* if bit $\lceil (i-2)/2 \rceil$ of L is 1, otherwise *stay.*
In rounds $i > 2\lfloor \log L \rfloor + 4$ behave as for round $1 + (i - 1 \bmod 2\lfloor \log L \rfloor + 4)$.

The following illustrates the execution of Procedure Extend-Labels for label 101:

1	0	1	1	0	0	1	1	1	0	1	1	0	0	1	1	1	0	···

Theorem 1. *Procedure Extend-Labels performs rendezvous on graph K_2 in at most $2\lfloor \log l \rfloor + 6$ rounds.*

Proof. Assume, without loss of generality, that agent 1 starts not later than agent 2. Rounds are counted from the startup of agent 2. The proof is divided into four cases.
Case 1. τ is odd. In this case, either agent 1 stays in the second round or a rendezvous is accomplished in one of the first two rounds. If agent 1 stays in the

second round, since this is an odd round for this agent, it also stays in the third round. In the third round, however, agent 2 moves, as its action corresponds to the most significant bit of L_2, which is 1. Thus rendezvous is accomplished no later than in round 3.

Case 2. τ is even and not divisible by $2\lfloor\log L_1\rfloor + 4$. In this case, the actions of agent 1 in the first two rounds are decided by the same bit in L_1. Thus, the actions of the two agents will be different in one of the rounds and a rendezvous is accomplished no later than in round 2.

Case 3. τ is even and divisible by $2\lfloor\log L_1\rfloor + 4$, and $\lfloor\log L_1\rfloor = \lfloor\log L_2\rfloor$. In this case, at least one bit must be different in both labels. Let b be the position of this bit. In round $2b+1$ the behavior of the two agents is different, and a rendezvous is accomplished. Thus, rendezvous is accomplished no later than in round $2\lfloor\log L_1\rfloor + 3$.

Case 4. τ is even and divisible by $2\lfloor\log L_1\rfloor + 4$, and $\lfloor\log L_1\rfloor \neq \lfloor\log L_2\rfloor$. In this case, the actions of the agent with the smaller label are different in rounds $2\lfloor\log l\rfloor + 5$ and $2\lfloor\log l\rfloor + 6$, while the other agent performs the same action. This results in a rendezvous no later than in round $2\lfloor\log l\rfloor + 6$. □

We can now formulate the following general algorithm for rendezvous in trees.

Algorithm Rendezvous-in-Trees

```
(1) Explore the tree
(2) if there is a central node then go to this node and stay;
    else go to one endpoint of the central edge;
            execute Procedure Extend-Labels;
```

Theorem 2. *Algorithm Rendezvous-in-Trees performs rendezvous on any n-node tree in $O(n + \log l)$ rounds. On the other hand, there exist n-node trees on which any rendezvous algorithm requires time $\Omega(n + \log l)$, even with simultaneous startup.*

Proof. The upper bound follows the fact that exploration of an n-node tree can be done in time $O(n)$. Now, consider an n-node tree, with $n = 2k$, consisting of two stars of degree $k-1$ whose centers are joined by an edge. Suppose that agents are initially located in centers of these stars and start simultaneously. The adversary can prevent an agent from finding the edge joining their initial positions for $2(k-1) \in \Omega(n)$ rounds. (After each unsuccessful attempt, the agent has to get back to its starting node.) This proves the lower bound $\Omega(n)$ on the time of rendezvous. In order to complete the proof, it is enough to show the lower bound $\Omega(\log l)$. We prove it in the simpler case of the two-node graph. (This also proves that Procedure Extend-Label is optimal for the two-node graph.) It is easy to extend the argument for our tree.

For any integer $x > 2$ and any rendezvous algorithm working in $t < x - 1$ rounds, we show two labels L_1 and L_2 of length x, such that the algorithm fails if agents placed on both ends of the edge have these labels. Let S_i be the binary sequence of length t describing the move/stay behavior of agent i (if the agent moves in round r, the rth bit of its sequence is 1, otherwise it is 0). Since S_i is a function of L_i, and there are only $2^t < 2^{x-1}$ possible sequences S_i, it follows that there exist two distinct labels L_1 and L_2 of length x, such that $S_1 = S_2$. Pick those two labels. During the first t rounds, agents exhibit the same move/stay behavior, and hence they cannot meet. □

3 Rendezvous in Rings

In this section we assume that agents know that the underlying graph is a ring, although, in general, they do not know its size.

3.1 Simultaneous Startup

We provide an algorithm that performs rendezvous on a ring in the simultaneous startup model, and prove that it works in an asymptotically optimal number of rounds. In order to simplify the presentation, we first propose two algorithms that work only under certain additional conditions, and then show how to merge these algorithms into the final algorithm, working in all cases. Our first algorithm, Similar-Length-Labels, works under the condition that the lengths of the two labels are similar, more precisely, $\lfloor \log\log L_1 \rfloor = \lfloor \log\log L_2 \rfloor$. Let the extended label L_i^*, be a sequence of bits of length $2^{\lfloor \log\log L_i \rfloor + 1}$, consisting of the binary representation of label L_i preceded by a (possibly empty) string of zeros. For example, the label 15 corresponds to the binary sequence 1111, while the label 16 corresponds to 00010000. The algorithm is formulated for an agent with label L and corresponding extended label L^*. Let $m = 2^{\lfloor \log\log L \rfloor + 1}$ be the length of L^*. The algorithm works in stages numbered $1, 2, \ldots$ until rendezvous is accomplished. Stage s consists of m phases, each of which has 2^{s+1} rounds.

Algorithm Similar-Length-Labels

```
In phase $b$ of stage $s$ do:
if bit $b$ of $L^*$ is 1 then
    (1) move for $2^{s-1}$ rounds in an arbitrary direction from the starting node;
    (2) move for $2^s$ rounds in the opposite direction;
    (3) go back to the starting node
else stay for $2^{s+1}$ rounds.
```

Lemma 1. *Algorithm Similar-Length-Labels performs rendezvous in $O(D \log l)$ rounds on a ring, if $\lfloor \log\log L_1 \rfloor = \lfloor \log\log L_2 \rfloor$.*

Proof. If $\lfloor \log\log L_1 \rfloor = \lfloor \log\log L_2 \rfloor$, the lengths of the extended labels of both agents are equal, and therefore any phase b of any stage s of agent 1 starts and ends at the same time as for agent 2. Since $L_1 \neq L_2$, one of the agents is moving while the other is staying in at least one phase b of every stage. During stage s, every node at distance 2^{s-1} from the starting point of the agent will be visited. Thus, in phase b of stage s, where s is the smallest integer such that $2^{s-1} \geq D$, the agent that moves in this phase meets the agent that stays in this phase. The number of rounds in this stage is $O(D \log l)$, which also dominates the sum of rounds in all previous stages. □

Our second algorithm, Different-Length-Labels, works under the condition that $\lfloor \log\log L_1 \rfloor \neq \lfloor \log\log L_2 \rfloor$. Let the *activity number* b_i for agent i be 1 if $L_i = 1$, and $2 + \lfloor \log\log L_i \rfloor$ otherwise. The algorithm is formulated for an agent with label L and activity number b. The algorithm works in stages. Stage s consists of s phases. Phase p of any stage consists of 2^{p+1} rounds.

Algorithm Different-Length-Labels
In stage $s < b$, stay;
In stage $s \geq b$, stay in phases $p \neq s - b + 1$;
In phase $p = s - b + 1$ of stage $s \geq b$, move for 2^{p-1} rounds in an arbitrary direction, move in the opposite direction for 2^p rounds, and go back to the starting node.

Lemma 2. *Algorithm Different-Length-Labels performs a rendezvous in $O(D \log l)$ rounds, if $\lfloor \log\log L_1 \rfloor \neq \lfloor \log\log L_2 \rfloor$.*

Proof. If $\lfloor \log\log L_1 \rfloor \neq \lfloor \log\log L_2 \rfloor$, the agents have different activity numbers. Assume, without loss of generality, that $l = L_1$ Hence $b_1 < b_2$. In stage $s \geq b_1$, agent 1 visits every node within distance 2^{s-b_1} from its starting node (in phase $s - b_1 + 1$). Hence, rendezvous is accomplished in stage s, where s is the smallest integer such that $2^{s-b_1} \geq D$, i.e., $s = b_1 + \lceil \log D \rceil$. Stage s consists of $O(2^s)$ rounds and dominates the sum of rounds in all previous phases. The required number of rounds is thus $O(2^{b_1 + \log D}) = O(D \log l)$. □

We now show how to combine Algorithm Similar-Length-Labels with Algorithm Different-Length-Labels into an algorithm that works for entirely unknown labels. The idea is to interleave rounds where Algorithm Similar-Length-Labels is performed with rounds where Algorithm Different-Length-Labels is performed. However, this must be done with some care, as an agent cannot successfully switch algorithms when away from its starting node. The solution is to assign slices of time of increasing size to the algorithms. At the beginning of a phase of each of the algorithms, the agent is at its starting node. If it can complete the given phase of this algorithm before the end of the current time slice, it does so. Otherwise it waits (at its starting node) until the beginning of the next time slice (devoted to the execution of the other algorithm), and then proceeds with the execution of the halted phase in the following time slice. (Note that, while one agent remains idle till the end of a time slice, the other agent might be active, if Algorithm Similar-Length-Labels is executed and the label lengths are in different ranges.)

It only remains to specify the sequence of time slices. Let time slice t consist of 2^{t+1} rounds (shorter slices than 4 rounds are pointless). It is now enough to notice that the phases up for execution during a time slice will never have more rounds than the total number of rounds in the slice. As a phase of an algorithm has never more than twice the number of rounds of the preceding phase, at least a constant fraction of every time slice is actually utilized by the algorithm. Exactly one of the algorithms has its condition fulfilled by the labels, and this algorithm accomplishes a rendezvous in $O(D \log n)$ rounds, while the other algorithm has been assigned at most twice as many rounds in total.

Theorem 3. *In the simultaneous startup model, the minimum time of rendezvous in the ring is $\Theta(D \log l)$.*

Proof. The upper bound has been shown above. For the lower bound, if $D = 1$, then the lower bound proof from the previous section is easy to modify for the ring. Thus assume that $D > 1$. We actually prove a lower bound for the weaker task *cross-or-meet* in which the two agents have either to meet at the same node, or to simultaneously traverse an edge in the two opposite directions. Clearly, an algorithm solving cross-or-meet in r rounds for two agents at distance D solves cross-or-meet in at most r rounds for two agents at distance $D - 1$. Thus we assume without loss of generality that D is even. Define an infinite sequence of consecutive segments of the ring, of $D/2$ vertices each, starting clockwise from an arbitrary node in the ring. Note that the starting nodes of the agents are located in two different segments, with one or two segments in between. Note also that the two agents have the same position within their segments. Divide all rounds into periods of $D/2$ rounds each, with the first round as the first round of the first period. During any period, an agent can only visit nodes of the segment where it starts the period and the two adjacent segments.

Suppose that port numbers (fixed by the adversary at every node) yield an orientation of the ring, i.e., for any node v, the left neighbor of the right neighbor of v is v. The behavior of an agent with label L, running algorithm A, yields the following sequence of integers in $\{-1, 0, 1\}$, called the *behavior code*. The tth term of the behavior code of an agent is -1 if the agent ends time period t in the segment to the left of where it began the period, 1 if it ends to the right and 0 if it ends in the segment in which it began the period. In view of the orientation of the ring, the behavior of an agent, and hence its behavior code, depends only on the label of the agent. Note that two agents with the same behavior code of length x, cannot accomplish cross-or-meet during the first x periods, if they start separated by at least one segment: even though they may enter the same segment during the period, there is insufficient time to visit the same node or the same edge.

Assume that there exists an algorithm A which accomplishes cross-or-meet in $Dy/6$ rounds. This time corresponds to at most $y/2$ periods. There are only $3^{y/2} < 2^y$ behavior codes of length $y/2$. Hence it is possible to pick two distinct labels L_1 and L_2 not greater than 2^y, for which the behavior code is the same. For these labels algorithm A does not accomplish cross-or-meet in $Dy/6$ rounds. This contradiction implies that any cross-or-meet algorithm, and hence any rendezvous algorithm, requires time $\Omega(D \log l)$. □

3.2 Arbitrary Startup

We begin by observing that, unlike in the case of simultaneous startup, $\Omega(n)$ is a natural lower bound for rendezvous time in an n-node ring, if startup is arbitrary, even for bounded distance D between starting nodes of the agents. Indeed, since starting nodes can be antipodal, each of the agents must at some point travel at distance at least $n/4$ from its starting node, unless he meets the other agent before. Suppose that the later agent starts at the time when the earlier agent is at distance $n/4$ from its starting node v. The distance D between the starting node of the later agent and v can be any number from 1 to an, where $a < 1/4$. Then rendezvous requires time $\Omega(n)$ (counting, as usual, from the startup of the later agent), since at the startup of the later agent the distance between agents is $\Omega(n)$. On the other hand, the lower bound $\Omega(D \log l)$ from the previous subsection is still valid, since the adversary may also choose simultaneous startup. Hence we have:

Proposition 1. *In the arbitrary startup model, the minimum time of rendezvous in the n-node ring is $\Omega(n + D \log l)$.*

We now turn attention to upper bounds on the time of rendezvous in the ring with arbitrary startup. Our next result uses the additional assumtion that the size n of the ring is known to the agents. The idea is to modify Procedure Extend-Labels. Every round in Procedure Extend-Labels is replaced by $2n$ rounds: the agent stays, respectively moves in one (arbitrary) direction, for this amount of time. Recall that in the Procedure Extend-Labels the actions of the two agents differ in round $2\lfloor \log l \rfloor + 6$ at the latest (counting from the startup of the later agent). In the modified procedure, time segments of activity or passivity, lasting $2n$ rounds, need not be synchronized between the two agents (if τ is not a multiple of $2n$) but these segments clearly overlap by at least n rounds. More precisely, after time at most $2n(2\lfloor \log l \rfloor + 6)$, there is a segment of n consecutive rounds in which one agent stays and the other moves in one direction. This must result in a rendezvous. Thus we have the following result which should be compared to the lower bound from Proposition 1. (Note that this lower bound holds even when agents know n.)

Theorem 4. *For a ring of known size n, rendezvous can be accomplished in $O(n \log l)$ rounds.*

The above idea cannot be used for rings of unknown size, hence we give a different algorithm working without this additional assumption. We first present the idea of the algorithm. Without loss of generality assume that $L_1 > L_2$. Our goal is to have agent 1 find agent 2 by keeping the latter still for a sufficiently long time, while agent 1 moves along the ring. Since agents do not know whose label is larger, we schedule alternating segments of activity and passivity of increasing length, in such a way that the segments of agent 1 outgrow those of agent 2. The algorithm is formulated for an agent with label L.

Algorithm Ring-Arbitrary-Startup

```
For $k = 1, 2, \ldots$ do
(1) Move for $kL$ rounds in one (arbitrary) direction;
(2) Stay for $kL$ rounds.
```

Theorem 5. *Algorithm Ring-Arbitrary-Startup accomplishes rendezvous in $O(l\tau + ln^2)$ rounds.*

Proof. Without loss of generality assume that $L_1 > L_2$. First suppose that agent 2 starts before agent 1. Agent 1 performs active and passive segments of length kL_1 from round $k(k-1)L_1 + 1$ to round $k(k+1)L_1$. The length of the time segment of agent 1, containing round t, is $\lfloor 1/2 + \sqrt{1/4 + (t-1)/L_1} \rfloor L_1$. Similarly, the length of the seqment of agent 2, containing round t, is $\lfloor 1/2 + \sqrt{1/4 + (t+\tau-1)/L_2} \rfloor L_2$. There exists a constant c such that after round cn^2 every passive segment of agent 2 is of length greater than n. It now remains to establish when the active segments of agent 1 are sufficiently longer than those of agent 2. When the difference is $2n$ or larger, there are at least n consecutive rounds where agent 1 moves (and thus visits every node of the ring), while agent 2 stays. In the worst case $L_1 = L_2 + 1 = l + 1$ and the inequality $\lfloor 1/2 + \sqrt{1/4 + (t-1)/(l+1)} \rfloor (l+1) - \lfloor 1/2 + \sqrt{1/4 + (t+\tau-1)/l} \rfloor l \geq 2n$ is satisfied by some $t \in O(l\tau + ln^2)$.

If agent 2 starts after agent 1, the condition that the length of the passive segments of agent 2 is of length at least n is still satisfied after round cn^2, for some constant c, and the second condition (concerning the difference between the agents' segments) is satisfied even sooner than in the first case.

Rendezvous is accomplished by the end of the segment containing round $t \in O(l\tau + ln^2)$. Since the length of this segment is also $O(l\tau + ln^2)$, this concludes the proof. □

In the above upper bound there is a factor l instead of $\log l$ from the simultaneous startup scenario. It remains open if l is a lower bound for rendezvous time in the ring with arbitrary startup.

4 Rendezvous in Arbitrary Connected Graphs

Simultaneous startup. For the scenario with simultaneous startup in arbitary connected graphs, we will use techniques from Section 3.1, together with the following lemma.

Lemma 3. *Every node within distance D of a node v in a connected graph of maximum degree Δ, can be visited by an agent, starting in v and returning to v, in $O(D\Delta^D)$ rounds.*

Proof. Apply breadth-first search. There are $O(\Delta^D)$ paths of length at most D originating in node v. Thus, in $O(D\Delta^D)$ rounds, all of these paths are explored and all nodes within distance D are visited. □

We keep the exact pattern of activity and passivity from the interleaving algorithm of Section 3.1 but replace the linear walk from the starting node by a breadth-first search walk: if alloted time in a given phase is t, the agent performs breadth-first search for $t/2$ rounds and then backtracks to the starting node. Since the only difference is that we now require a phase of length $O(D\Delta^D)$ to accomplish rendezvous, instead of a phase of length $O(D)$ for the ring, we get the following result.

Theorem 6. *Rendezvous can be accomplished in $O(D\Delta^D \log l)$ rounds in an arbitrary connected graph with simultaneous startup.*

Note that agents do not need to know the maximum degree Δ of the graph to perform the above algorithm. Also note that the above result is optimal for bounded distance D between agents and bounded maximum degree Δ, since $\Omega(\log l)$ is a lower bound.

Arbitrary startup. We finally show that rendezvous is feasible even in the most general situation: that of an arbitrary connected graph and arbitrary startup. The idea of the algorithm is to let the agent with smaller label be active and the agent with larger label be passive for a sufficiently long sequence of rounds to allow the smaller labeled agent to find the other. This is accomplished, as in the corresponding scenario for the ring, by an increasing sequence of time segments of activity and passivity. However, this time we need much longer sequences of rounds. The algorithm is formulated for an agent with label L.

```
Algorithm General-Graph-Arbitrary-Startup For $k = 1, 2, \ldots$ do
(1) Perform breadth-first search for $k10^L$ rounds;
(2) Stay for $k10^L$ rounds.
```

Theorem 7. *Algorithm General-Graph-Arbitrary-Startup accomplishes rendezvous.*

Proof. Without loss of generality assume that $L_1 > L_2$. First suppose that agent 2 starts before agent 1. There exists a positive integer t such that, after t rounds, we have: (1) the length of (active) segments of agent 2 is $> n^n$, and (2) length of (passive) segments of agent 1 is at least three times larger than the active (and passive) segments of agent 2. Statement 1 is obviously correct, since the lengths of the segments form an increasing sequence of integers. Statement 2 is true, since the ratio of the length of segments of agent 1 and the length of segments of agent 2 is $\sqrt{\frac{10^{L_1}t}{10^{L_2}(t+\tau)}} \geq \sqrt{\frac{10t}{(t+\tau)}} \geq 3$, for sufficiently large t. (This is the reason for choosing base 10 for time segments of length $k10^L$). Hence, after t rounds, two complete consecutive segments of agent 2 (one segment active and one segment passive) are contained in a passive segment of agent 1. Since the active segment of agent 2 is of size larger than n^n, this guarantees rendezvous. If agent 2 starts after agent 1, the above conditions are satisfied even sooner. □

Note that the argument to prove correctness of Algorithm Ring-Arbitrary-Startup cannot be directly used for arbitrary connected graphs. Indeed, in the general case, it is not sufficient to show that an arbitrarily large part of an active segment of one agent is included in a passive segment of the other. Instead, since breadth-first search is used, we require a stronger property: the inclusion of an entire active segment (or a fixed fraction of it). This, in turn, seems to require segments of size exponential in L. We do not know if this can be avoided.

5 Conclusion

The rendezvous problem is far from beeing completely understood even for rings. While for simultaneous startup, we established that optimal rendezvous time is $\Theta(D \log l)$, our upper bound on rendezvous time in rings for arbitrary startup contains a factor l, instead of $\log l$. It remains open if l is also a lower bound in this case. For arbitrary connected graphs we proved feasibility of rendezvous even with arbitrary startup but our rendezvous algorithm is very inefficient in this general case. The main open problem is to establish if fast rendezvous is possible in the general case. More specifically: question **Q2** from the introduction remains unsolved in its full generality.

Acknowledgements. Andrzej Pelc is supported in part by NSERC grant OGP 0008136 and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais. This work was done during the first and second authors visit at the Research Chair in Distributed Computing of the Université du Québec en Outaouais.

References

1. S. Alpern. The rendezvous search problem. SIAM J. on Control and Optimization 33(3), pp. 673–683, 1995.
2. S. Alpern. Rendezvous search on labelled networks. Naval Reaserch Logistics 49, pp. 256–274, 2002.
3. S. Alpern and S. Gal. The theory of search games and rendezvous. Int. Series in Operations research and Management Science, number 55, Kluwer Academic Publishers, 2002.
4. J. Alpern, V. Baston, and S. Essegaier. Rendezvous search on a graph. Journal of Applied Probability 36(1), pp. 223–231, 1999.
5. S. Alpern and S. Gal. Rendezvous search on the line with distinguishable players. SIAM J. on Control and Optimization 33, pp. 1270–1276, 1995.
6. E. Anderson and R. Weber. The rendezvous problem on discrete locations. Journal of Applied Probability 28, pp. 839–851, 1990.
7. E. Anderson and S. Essegaier. Rendezvous search on the line with indistinguishable players. SIAM J. on Control and Optimization 33, pp. 1637–1642, 1995.
8. E. Anderson and S. Fekete. Asymmetric rendezvous on the plane. Proc. 14th Annual ACM Symp. on Computational Geometry, 1998.
9. E. Anderson and S. Fekete. Two-dimensional rendezvous search. Operations Research 49, pp. 107–118, 2001.
10. V. Baston and S. Gal. Rendezvous on the line when the players' initial distance is given by an unknown probability distribution. SIAM J. on Control and Optimization 36, pp. 1880–1889, 1998.
11. V. Baston and S. Gal. Rendezvous search when marks are left at the starting points. Naval Res. Log. 48, pp. 722–731, 2001.
12. P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer, Gathering of asynchronous oblivious robots with limited visibility, Proc. 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2001), LNCS 2010, pp. 247–258, 2001.
13. S. Gal. Rendezvous search on the line. Operations Research 47, pp. 974–976, 1999.
14. J. Howard. Rendezvous search on the interval and circle. Operation research 47(4), pp. 550–558, 1999.
15. W. Lim and S. Alpern. Minimax rendezvous on the line. SIAM J. on Control and Optimization 34(5), pp. 1650–1665, 1996.
16. T. Schelling. The strategy of conflict. Oxford University Press, Oxford, 1960.
17. L. Thomas. Finding your kids when they are lost. Journal on Operational Res. Soc. 43, pp. 637–639, 1992.