Polynomial Deterministic Rendezvous in Arbitrary Graphs

Dariusz R. Kowalski[1,2] and Andrzej Pelc[3,⋆]

[1] Max-Planck-Institut für Informatik,
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
darek@mpi-sb.mpg.de

[2] Instytut Informatyki, Uniwersytet Warszawski,
Banacha 2, 02-097 Warszawa, Poland

[3] Département d'informatique, Université du Québec en Outaouais,
Hull, Québec J8X 3X7, Canada
Andrzej.Pelc@uqo.ca

Abstract. The rendezvous problem in graphs has been extensively studied in the literature, mainly using a randomized approach. Two mobile agents have to meet at some node of a connected graph. We study deterministic algorithms for this problem, assuming that agents have distinct identifiers and are located in nodes of an unknown anonymous connected graph. Startup times of the agents are arbitrarily decided by the adversary. The measure of performance of a rendezvous algorithm is its *cost*: for a given initial location of agents in a graph, this is the number of steps since the startup of the later agent until rendezvous is achieved. Deterministic rendezvous has been previously shown feasible in arbitrary graphs [16] but the proposed algorithm had cost *exponential* in the number n of nodes and in the smaller identifier l, and polynomial in the difference τ between startup times. The following problem was stated in [16]: Does there exist a deterministic rendezvous algorithm with cost polynomial in n, τ and in labels L_1, L_2 of the agents (or even polynomial in n, τ and $\log L_1$, $\log L_2$)? We give a positive answer to both problems: our main result is a deterministic rendezvous algorithm with cost polynomial in n, τ and $\log l$. We also show a lower bound $\Omega(n^2)$ on the cost of rendezvous in some family of graphs.

1 Introduction

Two mobile agents located in nodes of an undirected connected graph, have to meet at some node of the graph. This task is known in the literature as the rendezvous problem in graphs, and in this paper we study deterministic algorithms to solve it efficiently. 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, if the graph models an unknown environment, a unique labeling of nodes may not be available, or agents may be unable to recognize node labels. Hence it is important to design rendezvous algorithms for

⋆ Research supported in part by NSERC grant and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais.

agents operating in *anonymous* graphs, i.e., graphs without unique labeling of nodes. Clearly, the agents must be capable of *locally* distinguishing ports at a node: otherwise, it may even be impossible to visit all neighbors of a node of degree 3 (after visiting the second neighbor, it is impossible to distinguish the port leading to the first visited neighbor from that leading to the unvisited one). Consequently, agents initially located at two such nodes, 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 local port labelings is assumed. We do not assume any knowledge of the topology of the graph, of its size, or of the distance separating the agents.

1.1 The Model

Synchrony and Startup Times. Agents move in synchronous steps. In every step, an agent may either remain in the same node or move to an adjacent node. We assume that startup times of the agents are arbitrarily decided by an adversary. Agents are not aware of the difference between startup times, and each of them starts executing the rendezvous algorithm and counting steps 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 rather than waiting in the node before it.

Adversarial Decisions and Cost of Rendezvous. An agent, currently located at a node, is not aware of 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. If agents get to the same node in the same round, they become aware of it and rendezvous is achieved. The *cost* of a rendezvous algorithm, for a given initial location of agents in a graph, is the worst-case number of steps 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. In particular, time of local computations performed by agents does not contribute to cost.

Labels and Local Knowledge. 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 of simultaneously starting agents in a two-node graph. Hence we assume that agents have distinct *labels*, which are two different integers, and that every agent knows its own label. (For technical reasons we assume that labels are larger than 1. This assumption can be easily omitted.) If *both* agents knew *both* labels, the problem could 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. (This strategy is sometimes called "wait for mommy".) However, the assumption that agents know each other may often be unrealistic, as they may be created in different parts of the network 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.

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 the value of its label). Labels are distinct integers larger than 1. l denotes the smaller of the two labels. The difference between startup times of the agents is denoted by τ. We use the word "graph" to mean a simple undirected connected graph with local port labelings but without node labels. n denotes the number of nodes in the graph.

1.2 Our Results

In [16], deterministic rendezvous was considered under the above described model. The authors formulated the following two questions:

Q1. Is rendezvous feasible in arbitrary graphs?
Q2. If so, can it be performed in cost polynomial in n, τ, L_1 and L_2 (or even polynomial in n, τ, $\log L_1$ and $\log L_2$)?

They gave an affirmative answer to the first question but their rendezvous algorithm had cost exponential in n and l (and polynomial in τ). The second question was left open.

We give a positive answer to both versions of this question. Our main result is a deterministic rendezvous algorithm with cost polynomial in n, τ and $\log l$. The algorithm contains a non-constructive ingredient: agents use combinatorial objects whose existence we prove by the probabilistic method. Nevertheless our algorithm is indeed deterministic. Both agents can find separately the same combinatorial object with desired properties (which is then used in the rendezvous algorithm). This can be done using brute force exhaustive search which may be quite complex but in our model only moves of the agents are counted and computation time of the agents does not contribute to cost. Moreover, it should be noticed that finding this combinatorial object can be done a single time at a preprocessing stage, the object can be stored in agents' memory and subsequently used in many instances of the rendezvous problem. We also show a lower bound $\Omega(n^2)$ on rendezvous cost in some family of graphs.

The paper is organized as follows. In Section 2 we construct a simpler rendezvous algorithm polynomial in n, τ and l (instead of $\log l$). We do this to first present the main idea of the algorithm and of its analysis without additional complications needed to decrease the cost. In Section 3 we show how to modify this algorithm, in order to decrease its cost to polynomial in n, τ and $\log l$. In Section 4 we establish the lower bound $\Omega(n^2)$ on rendezvous cost in some family of graphs. Section 5 contains conclusions and open problems.

1.3 Related Work

The rendezvous problem has been introduced in [23]. The vast body of results on rendezvous (see the book [4] 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., [11, 12, 19], or in the plane, see, e.g., [9, 10]), and those discussing rendezvous in graphs, e.g., [2, 5]. Most of the papers, e.g., [2, 3, 7, 11, 20] consider the probabilistic scenario: inputs and/or rendezvous strategies are random. In [20] randomized rendezvous strategies are applied to study self-stabilized token management schemes. Randomized rendezvous strategies use random walks in graphs, which have been widely studied and applied also, e.g., in graph traversing [1], on-line algorithms [14] and estimating volumes of convex bodies [17]. A natural extension of the rendezvous problem is that of gathering [18, 20, 22, 24], when more than 2 agents have to meet in one location.

Deterministic rendezvous with anonymous agents working in unlabeled graphs but equipped with tokens used to mark nodes was considered e.g., in [21]. In [25] the authors considered rendezvous of many agents with unique labels. Although one of their scenarios is deterministic, it differs from our setting in that agents know the graph and they know a finite set containing the team of agents that are supposed to meet. Deterministic rendezvous in unlabeled graphs, assuming that each agent knows only its own identity, was considered in [16]. The authors considered rendezvous under the scenario adopted in the present paper, and under another scenario which additionally assumed simultaneous startup. They gave efficient rendezvous algorithms for trees and rings and proved feasibility of rendezvous for arbitrary graphs. In the case of arbitrary startup times (which we assume in the present paper) their algorithm for arbitrary graphs was *exponential* in n and l (and polynomial in τ).

2 A Rendezvous Algorithm Polynomial in n, τ and l

2.1 Deterministic Polynomial Covering of a Graph

A *walk* of length k in a graph is a sequence $(v_1, ..., v_k)$ of nodes such that node v_{i+1} is adjacent to v_i, for all $i < k$. A *covering walk* is a walk in which every node of the graph appears at least once. The aim of this subsection is to give a deterministic procedure, using a number of steps polynomial in n which, when started in any node of an unknown graph with at most n nodes, produces a covering walk in this graph. This procedure will be an important ingredient in our rendezvous algorithms.

Define a *random walk* of an agent in graph G as a walk in which the agent, currently located at a node of degree d, acts in the next step as follows: it remains in the node with probability $1/2$ and moves through any port with the same probability $1/(2d)$. The *cover time* of the graph G during a random walk starting at node v is the random variable denoting the smallest number of steps after which the agent performing this walk visits all nodes of the graph. The *meeting time* of two agents performing simultaneous random walks in graph G, starting at nodes v and w, is the random variable denoting the smallest number of steps after which agents performing these walks meet at some node.

We will use the following Lemma proved in [15]:

Lemma 1. *There exists a constant $\alpha > 0$ such that the probability of each of the following events is at least $1/2$:*
Event E_1: the cover time of graph G with n nodes during the random walk starting at any node, is at most αn^3.
Event E_2: the meeting time of two agents performing simultaneous random walks in graph G with n nodes, is at most αn^3, for any starting nodes.

Let α be the constant from Lemma 1. Let $\lambda(n) = \lceil 2\alpha n^5 \log n \rceil$. The next lemma shows a useful property of a random walk (the proof is omitted).

Lemma 2. *A random walk of length $\lambda(n)$ starting at node v in a graph G with at most n nodes is a covering walk, with probability at least $1 - 2^{-2n^2 \log n}$.*

For any positive integer n and any function $h_n : \{1, ..., \lambda(n)\} \times \{1, ..., n-1\} \longrightarrow \{0, 1, ..., n-1\}$, such that $h_n(i,d) \leq d$, we define the following procedure describing a walk of length $\lambda(n)$ in a graph G, starting at a node v (cf. the upper bound for the length of a universal traversal sequence [1]).

Procedure GraphCover(n, h_n)
In step $i \leq \lambda(n)$, the agent, currently located at a node of degree d, moves to an adjacent node by port $h_n(i,d)$, or remains idle if $h_n(i,d) = 0$. After step $\lambda(n)$ it stops.

Lemma 3. *For any n, there exists a function $h_n : \{1, ..., \lambda(n)\} \times \{1, ..., n-1\} \longrightarrow \{0, 1, ..., n-1\}$, such that $h_n(i,d) \leq d$ and the Procedure* GraphCover(n, h_n) *starting at any node of any graph G with at most n nodes, produces a covering walk in this graph.*

Proof. Fix n. Fix a graph G with at most n nodes and fix some starting node v in G. We can do it in at most $n^{n^2} \cdot n \leq 2^{n^2(\log n + 1)}$ different ways, for fixed n. Applying Lemma 2, the probability of the event 'there exists a graph G with at most n nodes and a starting node v in G, such that the random walk of length $\lambda(n)$ in graph G starting in v is not a covering walk' is at most

$$2^{-2n^2 \log n} \cdot 2^{n^2(\log n + 1)} \leq 2^{-n} .$$

Using the probabilistic argument we prove the existence of the desired function, which completes the proof. □

Note that the problem of construction of function h_n satisfying Lemma 3 is hard (cf. hardness of a construction of a universal traversal sequence even for 3-regular graphs [13]).

In our applications to rendezvous algorithms, agents will use Procedure GraphCover(n, h_n) producing a covering walk in any graph with at most n nodes. To this end we want each of the agents to find the same function h_n whose existence is guaranteed by Lemma 3. (This can be done by exhaustive search, ordering all such possible functions in a canonical way and checking them one by one to find the first suitable one. Recall that, according to our model, only moves of the agents are accounted for, and computation time of the agents does not contribute to rendezvous cost.) Let $\hat{h}_n$ be the first function in this canonical ordering, satisfying Lemma 3, for any n. To simplify notation, we will write GraphCover(n) instead of GraphCover$(n, \hat{h}_n)$, throughout the paper.

2.2 Construction and Analysis of Rendezvous Algorithm PA

In order to design rendezvous algorithm PA, we will use procedure GraphCover(n), which takes $\lambda(n) = \lceil 2\alpha n^5 \log n \rceil$ steps and produces a covering walk in any graph with at most n nodes.

We will show that the following algorithm completes rendezvous in any n-node graph, for agents with arbitrary labels L_1, L_2, with arbitrary delay τ, in cost polynomial in n, $l = \min\{L_1, L_2\}$ and τ.

Algorithm PA (PassiveActive) for agent with label L.

```
For $k = 1, 2, \ldots$ do
    Passive Phase: Wait for $2Lk$ steps
    Active Phase:
        – Perform GraphCover($Lk$), starting from the current node in the graph
        – Perform $L$ times GraphCover($k$), always starting from the current node in the graph
```

Let $k_0 = \lambda(n)$. The idea of the algorithm is to guarantee that one of the agents is passive while the other agent L performs GraphCover(k_0) and thus completes rendezvous. (We refer to this situation by saying that the active agent meets the passive agent – an asymmetric relation.) This is the reason for having increasing time segments of activity and passivity. The turn of the "for" loop for a given k will be called the kth *epoch* of the agent. The kth epoch of agent with label L has two *phases* of equal length $2Lk$: the passive phase and the active phase. The active phase is composed of an execution of GraphCover(Lk) followed by L executions of GraphCover(k). This is the subtle point in the algorithm design: it seems that none of these parts alone (one long execution of GraphCover or many short executions of it) permits to guarantee rendezvous in cost polynomial in n, l and τ.

We analyze the performance of algorithm PA as a function of n, l and τ. Let G be an n-node graph and L_1, L_2 the labels of agents. Without loss of generality assume that $L_1 > L_2 = l$. We start counting time steps from the startup of the later agent. For every step t denote by $k_i(t)$, for $i = 1, 2$, the number of epoch executed by agent i in step t. We will use the following fact, which follows from the Properties of GraphCover(k), and from the definition of k_0.

Fact 1 **1.** *If one agent starts its active phase of epoch k in time t, where $L_1 k \geq 2k_0$, and the other agent is in the passive phase during the time segment $[t, t + k_0)$, then rendezvous is completed by step $t + k_0$.*

2. *Assume $k \geq k_0$. If one agent is in the second half of its active phase of epoch k in the time segment $[t, t + 2k_0)$, and the other agent is in a passive phase during the time segment $[t, t + 2k_0)$, for some t, then rendezvous is completed by step $t + 2k_0$.*

3. *Assume $k \geq k_0$. If one agent ends its active phase of epoch k in time t, and the other agent is in a passive phase during the time segment $(t - k_0, t]$, then rendezvous is completed by step t.*

The following lemma estimates cost of rendezvous under some technical conditions (the proof will appear in the full version of the paper).

Lemma 4. *Let t_1, t_2 be steps ending epochs $k_1(t_1), k_2(t_2)$ of the first and second agent, respectively. Assume that $L_1k_1(t_1) \geq 40k_0$, $k_2(t_2) \geq 10k_0$, $|t_1 - t_2| \leq 4k_0$ and $|L_1k_1(t_1) - L_2k_2(t_2)| \leq 2k_0$. Then rendezvous is completed by step $t_2 + 26lk_2(t_2)k_0$.*

The next three lemmas estimate the time step by which rendezvous is completed, depending on the number of epoch changes of one agent during one epoch of the other (their proofs will appear in the full version of the paper).

Lemma 5.

1. *Let t be the beginning of epoch $k_1(t)$ of the first agent and assume that $L_1k_1(t) \geq 40k_0$ and $k_2(t) \geq 10k_0$. If the second agent ends its epoch only once during epoch $k_1(t)$ of the first agent, then rendezvous is completed by step $t + 11lk_2(t) + 26lk_2(t)k_0$.*

2. *Let t be the beginning of epoch $k_2(t)$ of the second agent and assume that $L_1k_1(t) \geq 40k_0$ and $k_2(t) \geq 10k_0$. If the first agent ends its epoch only once during epoch $k_2(t)$ of the second agent, then rendezvous is completed by step $t + 11lk_2(t) + 26lk_2(t)k_0$.*

Lemma 6.

1. *Let t be the beginning of epoch $k_1(t)$ of the first agent and assume that $L_1k_1(t) \geq 40k_0$ and $k_2(t) \geq 10k_0$. If the second agent ends its consecutive epochs twice during epoch $k_1(t)$ of the first agent, then rendezvous is completed by step $t + 14lk_2(t) + 26lk_2(t)k_0$.*

2. *Let t be the beginning of epoch $k_2(t)$ of the second agent and assume that $L_1k_1(t) \geq 40k_0$ and $k_2(t) \geq 10k_0$. If the first agent ends its consecutive epochs twice during epoch $k_2(t)$ of the second agent, then rendezvous is completed by step $t + 14lk_2(t) + 26lk_2(t)k_0$.*

Lemma 7.

1. *Let t be the beginning of epoch $k_1(t)$ of the first agent and assume that $L_1k_1(t) \geq 40k_0$ and $k_2(t) \geq 10k_0$. If the second agent ends its consecutive epochs at least three times during epoch $k_1(t)$ of the first agent, then rendezvous is completed by the end of epoch $k_2(t) + 1$ of the second agent, which is at most $t + 9lk_2(t)$.*

2. *Let t be the beginning of epoch $k_2(t)$ of the second agent and assume that $L_1k_1(t) \geq 40k_0$ and $k_2(t) \geq 10k_0$. If the first agent ends its consecutive epoch at least three times during epoch $k_2(t)$ of the second agent, then rendezvous is completed by the end of epoch $k_1(t) + 1$ of the first agent, which is at most $t + 4lk_2(t)$.*

Theorem 1. *Algorithm PA solves the rendezvous problem for any n-node graph G, for any labels $L_1 > L_2 = l$ of agents and any delay τ between startup times, in cost $\mathcal{O}(\sqrt{l\tau}n^5 \log n + ln^{10} \log^2 n)$.*

Proof. Let t_1 be the first step for which $L_1k_1(t_1) \geq 40k_0$. Let $t_2 \geq t_1$ be the first step for which $k_2(t_2) \geq 10k_0$. Observe that t_i is the beginning of epoch $k_i(t_i)$ of the ith agent. Consider the step $t^* = t_2 + (t_2 - t_1) + 8L_2k_2(t_2) = 2t_2 + 8L_2k_2(t_2) - t_1$. We have $t^* > t_2 \geq t_1$, hence $k_1(t_1) \leq k_1(t^*)$. Consider two cases.

Case A. $k_1(t_1) = k_1(t^*)$.

We have the inclusion $[t_2, t_2 + 4L_2k_2(t_2)] \subseteq [t_1, (t_1 + t^*)/2]$. Hence the epoch $k_2(t_2)$ of the second agent is included in the passive phase of epoch $k_1(t_1)$ of the first agent. We use Fact 1 point 1 to obtain that rendezvous is completed during epoch $k_2(t_2)$ of the second agent. Hence rendezvous cost is $\mathcal{O}(t_2 + L_2k_2(t_2))$. By definition of t_1 we get that $t_1 \in \mathcal{O}(k_0^2) = \mathcal{O}(n^{10}\log^2 n)$.

If $t_2 > t_1$ then $k_2(t_1) < 10k_0$, and consequently $k_2(t_2) = 10k_0$. Hence $t_2 = \mathcal{O}(L_2k_0^2) \in \mathcal{O}(ln^{10}\log^2 n)$. On the other hand, $L_2k_2(t_2) \in \mathcal{O}(ln^5 \log n)$.

If $t_2 = t_1$ then $t_2 \in \mathcal{O}(n^{10}\log^2 n)$. On the other hand we have

$$4L_2(k_2(1)+1) + \ldots + 4L_2(k_2(t_2)-1) \leq t_2 \ .$$

Hence $k_2(t_2) - k_2(1) \in \mathcal{O}(\sqrt{t_2/L_2})$. It follows that $L_2k_2(t_2) \in \mathcal{O}(L_2(k_2(1) + \sqrt{t_2/L_2}))$. Since $\tau \in \Omega(L_2(k_2(1))^2)$, we get that $k_2(1) \in \mathcal{O}(\sqrt{\tau/L_2})$, and hence rendezvous cost is

$$\mathcal{O}(t_2+L_2k_2(t_2)) \subseteq \mathcal{O}(t_2+L_2\sqrt{\tau/L_2}+L_2\sqrt{t_2/L_2}) \subseteq \mathcal{O}(n^{10}\log^2 n+\sqrt{l\tau}+\sqrt{l}n^5\log n).$$

Consequently, in both situations, rendezvous cost is $\mathcal{O}(ln^{10}\log^2 n + \sqrt{l\tau})$.

Case B. $k_1(t_1) < k_1(t^*)$.

In this case we have that $t_2 \in \mathcal{O}(k_0^2 l)$ and $k_2(t_2) \in \mathcal{O}(\sqrt{\tau/l} + k_0)$. Below we give the proof of this statement in all possible situations.

- $L_1k_1(1) \geq 40k_0$ and $k_2(1) \geq 10k_0$. In this case $t_2 = t_1 = 1$. We also have $\tau \in \Omega(L_2(k_2(1))^2)$, and consequently $k_2(t_2) = k_2(1) \in \mathcal{O}(\sqrt{\tau/l})$.
- $L_1k_1(1) \geq 40k_0$ and $k_2(1) < 10k_0$. In this case $t_1 = 1$ and $t_2 \in \mathcal{O}(L_2k_0^2) = \mathcal{O}(lk_0^2)$. Also $k_2(t_2) \in \mathcal{O}(k_0)$.
- $L_1k_1(1) < 40k_0$ and $k_2(1) \geq 10k_0$. In this case $L_1(k_1(t_1) - 1) < 40k_0$, which implies that $t_1 \in \mathcal{O}(L_1(k_1(t_1))^2) \subseteq \mathcal{O}(k_0^2/L_1)$. Also $t_2 = t_1$, which gives $t_2 \in \mathcal{O}(k_0^2)$. On the other hand, $\tau \in \Omega(L_2(k_2(1))^2)$, and consequently $k_2(t_2) \in k_2(1) + \mathcal{O}(k_0) \subseteq \mathcal{O}(\sqrt{\tau/l} + k_0)$.
- $L_1k_1(1) < 40k_0$ and $k_2(1) < 10k_0$. In this case $L_1(k_1(t_1) - 1) < 40k_0$, which implies that $t_1 \in \mathcal{O}(L_1(k_1(t_1))^2) \subseteq \mathcal{O}(k_0^2/L_1)$. Also $t_2 \in t_1 + \mathcal{O}(L_2k_0^2) = \mathcal{O}(lk_0^2)$. On the other hand, $k_2(t_2) \leq k_2(t_1) + 10k_0 \in k_2(1) + \mathcal{O}(k_0) = \mathcal{O}(k_0)$.

Let t be the first step after t_2 in which an epoch of the first agent starts. Notice that, by the assumption $k_1(t_1) < k_1(t^*)$, we have $t \leq t_2 + t^*$, and consequently $t \in \mathcal{O}(t_2 + L_2k_2(t_2))$. Hence $k_2(t) \in \mathcal{O}(k_2(t_2))$. Consider times $t_1', t_2' > t$ such that t_i' is the end of epoch $k_i(t)$, for $i = 1, 2$.

Subcase B1. $t_1' \leq t_2'$.
Consider epoch $k_2(t)$ of the second agent. By definition of step t, this epoch starts not earlier than t_2. Since $t_1' \leq t_2'$, we have that the first agent ends its epoch at least once during epoch $k_2(t)$ of the second agent. Applying point 2 of one of the Lemmas 5, 6 and 7, depending on the number of epoch changes of the first agent in epoch $k_2(t_2)$, we obtain that rendezvous cost is at most

$$\begin{aligned} &t + 18lk_2(t) + 26lk_2(t)k_0 \in \mathcal{O}(t_2 + L_2k_2(t_2)) + \mathcal{O}(lk_2(t_2)) + \mathcal{O}(lk_2(t_2)k_0) \subseteq \\ &\subseteq \mathcal{O}(k_0^2 l + l(\sqrt{\tau/l} + k_0) + l(\sqrt{\tau/l} + k_0)k_0) = \mathcal{O}(\sqrt{l\tau}n^5\log n + ln^{10}\log^2 n) \ . \end{aligned}$$

Subcase B2. $t'_1 > t'_2$.
Consider epoch $k_1(t)$ of the first agent. It starts in step $t \geq t_2$. Since $t'_1 > t'_2$, we have that the second agent ends its epoch at least once during epoch $k_1(t)$ of the first agent. Applying point 1 of one of the Lemmas 5, 6 or 7, depending on the number of epoch changes of the second agent in epoch $k_1(t_1)$, we obtain that rendezvous cost is at most

$$t + 18lk_2(t) + 26lk_2(t)k_0 \in \mathcal{O}(t_2 + L_2k_2(t_2)) + \mathcal{O}(lk_2(t_2)) + \mathcal{O}(lk_2(t_2)k_0) \subseteq$$
$$\subseteq \mathcal{O}(lk_0^2 + l(\sqrt{\tau/l} + k_0) + l(\sqrt{\tau/l} + k_0)k_0) \subseteq \mathcal{O}(\sqrt{l\tau}n^5 \log n + ln^{10} \log^2 n) \ ,$$

the same asymptotic bound as in Subcase B1. □

3 A Rendezvous Algorithm Polynomial in n, τ and $\log l$

In this section we design and analyze a modification of Algorithm PA which works in cost polynomial in n, τ and $\log l$, rather than polynomial in n, τ and l. The modified algorithm has two non-constructive ingredients: the function determining the covering walk, already used in Procedure GraphCover(n) and another one, used in the new procedure TRAVERSE described below. (As before, the new combinatorial object (a family of functions) whose existence we prove using again the probabilistic method, can be found by each of the agents separately, using local exhaustive search.) Similarly as before, our algorithm remains deterministic.

Assume that, for every label L and positive integer k, we have a function $f_{L,k}$: $\{1,...,k\lceil\log L\rceil\} \times \mathbb{Z}^+ \rightarrow \mathbb{Z}^+ \cup \{0\}$ such that $f_{L,k}(i,d) \leq d$ for any positive integers i, d. We call such a function a *port-function.* The interpretation of $f_{L,k}(i,d)$ is the port number used by agent with label L in the ith step of graph traversal with parameter k, if the agent is currently at a node of degree d (the value 0 indicates that the agent remains at the current node). According to this intuition we define the procedure TRAVERSE. For a non-negative integer t and for a positive integer k, define $T(k,t)$ as the set of all infinite sequences $(t_1, \ldots, t_k, \ldots)$ of non-negative integers such that $t_1 + \ldots + t_k = t$ and $t_i = 0$ for every $i > k$. For a given label L, integers $k > 0$ and $t \geq 0$, fix $\bar{t} \in T(k,t)$ and define:

```
Procedure TRAVERSE($L, k, \bar{t}$)
For $i = 1, 2, \ldots, k$ do initialize $count_i := t_i$ ($t_i$ is the $i$th value of $\bar{t}$)
For $j = 1, 2, \ldots, k\lceil\log L\rceil$ do
    Set $d$ to the degree of the current node (if $d > k$ and $count_d$ not initialized
        then initialize $count_d := 0$)
    Set $count_d := count_d + 1$
    If $f_{L,k}(count_d, d) > 0$ then Go using port $f_{L,k}(count_d, d)$
```

The intuition behind the parameter $\bar{t}$ in the above procedure is the following. For every d, we suppose that, before starting procedure TRAVERSE, t_d first port choices in nodes of degree d, yielded by the function $f_{L,k}$, were already executed. Hence this introduces a shift of procedure TRAVERSE by t steps back, assuming that t_d nodes of degree d were already visited. In the algorithm we will only use TRAVERSE for parameter $\bar{0}$ (hence no shift at all) but in the analysis we will consider executions of TRAVERSE shifted in time with respect to each other, and hence this more general formulation of the procedure will become useful.

The following algorithm uses procedure TRAVERSE, which in turn depends on a family of port-functions. We will use the algorithm for such a family of functions with a specially defined property.

```
Algorithm MPA (Modified PassiveActive) for agent with label $L$.
For $k = 1, 2, \ldots$ do
    Passive Phase: Wait for $2 \cdot 2\lceil \log L \rceil k$ steps
    Active Phase:
        First Stage: Perform GraphCover($\lceil \log L \rceil k$), starting from the current node in the graph
        Middle Stage: Perform TRAVERSE($L, k, \bar{0}$)
        Last Stage: Perform $2\lceil \log L \rceil$ times GraphCover($k$), always starting from the current node in the graph
```

3.1 Choosing Port-Functions

As before, the turn of the "for" loop for a given k will be called the kth epoch of the agent. Let G be an n-node graph, v_1, v_2 two nodes in G, t a non-negative integer, $ind \in \{1, 2\}$, $\bar{t} \in T(k_1, t)$, if $ind = 1$ and $\bar{t} \in T(k_2, t)$ otherwise. Execute($L_1, k_1, L_2, k_2, G, v_1, v_2, \bar{t}, ind$), denotes the execution of procedures

- TRAVERSE($L_1, k_1, \bar{t}$) and TRAVERSE($L_2, k_2, \bar{0}$) if $ind = 1$
- TRAVERSE($L_1, k_1, \bar{0}$) and TRAVERSE($L_2, k_2, \bar{t}$) if $ind = 2$

by agents operating in graph G, where procedure TRAVERSE($L_1, k_1, \cdot$) starts in node v_1 and procedure TRAVERSE($L_2, k_2, \cdot$) starts in node v_2. We assume that procedure Execute is performed until one of the agents completes its procedure TRAVERSE.

Let $\alpha > 0$ be the constant from Lemma 1 and let $k_0 = \lceil 2\alpha n^5 \log n \rceil$ be as in Section 2. We say that a family of port-functions $\{f_{L,k} : k \in \mathbb{Z}^+, L = 2, 3, \ldots\}$ is a *rendezvous family*, if the following property is satisfied:

RV for all labels $L_1 > L_2$ such that $\lceil \log L_1 \rceil = \lceil \log L_2 \rceil$, all parameters $k_1 = k_2$, for every n such that $10\alpha n^5 \log n \leq k_1$, every n-node graph G, for all starting nodes v_1, v_2, any sequence $\bar{t} \in T(k_1, t)$, where $t < 6k_0$, any $ind \in \{1, 2\}$, agents with labels L_1, L_2 meet during procedure Execute($L_1, k_1, L_2, k_2, G, v_1, v_2, \bar{t}, ind$).

Recall that, in view of Lemma 3 and of the choice of α, procedure GraphCover(n), lasting k_0 steps, produces a covering walk in any graph G with n nodes.

Our goal is to show the existence of a rendezvous family. The proof uses the probabilistic method and requires the analysis of simultaneous random walks of two agents in a graph. The proofs of the next two lemmas will appear in the full version of the paper.

Lemma 8. *Let L be a positive integer. For a given n-node graph G, consider two simultaneous random walks of length $10\alpha n^5 \log n \log L$, started in any nodes v_1, v_2 of graph G. The agents meet in some node with probability at least $1 - 2^{-10n^2 \log n \log L}$.*

Lemma 9. *There exists a rendezvous family of port-functions $\{f_{L,k} : k \in \mathbb{Z}^+, L = 2, 3, \ldots\}$.*

3.2 Analysis of Algorithm MPA

In view of Lemma 9 we can use a rendezvous family of port-functions as a basis for algorithm MPA. The rest of our analysis assumes that MPA uses such a fixed family (which agents can compute locally), and hence we assume that property RV is satisfied. Notice that, in the proof of Lemma 9, we used the probabilistic method for fixed k, L and n. The function $f_{L',k}$ of an agent with label L', such that $L/2 < L' \leq L$, has the domain bounded by $nk \log L$ and the range bounded by n. The agent may have to compute all such functions $f_{L'',k}$, for $L/2 < L'' \leq L$. This can be done locally in time exponential in $nk \log L \log n$. Recall that local computations do not affect rendezvous cost in our model.

Our next lemma corresponds to Lemma 4 in the analysis of algorithm PA. The proof will appear in the full version of the paper.

Lemma 10. *Let t_1, t_2 be steps ending epochs $k_1(t_1), k_2(t_2)$ of the first and second agent, respectively. Assume that $\lceil \log L_1 \rceil = \lceil \log L_2 \rceil$, $|\lceil \log L_1 \rceil k_1(t_1) - \lceil \log L_2 \rceil k_2(t_2)| < 2k_0$, $k_1(t_1), k_2(t_2) \geq 10\alpha n^5 \log n$ and $|t_1 - t_2| < 6k_0$. Then rendezvous is completed by step t_2.*

Theorem 2. *Algorithm MPA solves the rendezvous problem for any n-node graph G, for any labels $L_1 > L_2 = l$ of agents and for any delay τ between startup times, in cost $\mathcal{O}(n^5 \sqrt{\tau \log l} \log n + n^{10} \log^2 n \log l)$.*

The proof of Theorem 2 will appear in the full version of the paper.

4 A Lower Bound

The sharpest lower bound for deterministic rendezvous proved in [16] was $\Omega(n \log l)$. More precisely, the fact that rendezvous sometimes requires this cost follows from the lower bound $\Omega(D \log l)$ proved in [16] for agents starting at distance D in a ring. We show that, in some graphs with $\Theta(n^2)$ edges, the cost of rendezvous is $\Omega(n^2)$, i.e., a large part of the graph has to be explored before agents can meet. The proof will appear in the full version of the paper.

Theorem 3. *For all positive integers n and any labels L_1 and L_2, there exists an n-node graph G_n such that rendezvous cost in G_n for agents with labels L_1 and L_2 is $\Omega(n^2)$.*

5 Conclusion

Our main result was the design and analysis of a deterministic rendezvous algorithm polynomial in n, τ and $\log l$. This answers affirmatively question Q2 from [16]. Our algorithm requires exhaustive local search by each agent to find an object whose existence is proved using the probabilistic method. While local computations (even possibly very extensive), do not affect cost in our model, it is interesting to know if there is a deterministic rendezvous algorithm with cost polynomial in n, τ, $\log l$ whose local computations also take polynomial time.

Another open problem concerns the dependence of rendezvous cost on the parameter τ (the difference between startup times). We showed a lower bound

$\Omega(n^2)$ on rendezvous cost in some graphs. It was shown in [16] that cost $\Omega(\log l)$ is required even for the two-node graph. It was also shown in [16] that, for agents starting at distance $\Omega(n)$ in a ring, cost $\Omega(n \log l)$ is required, even for $\tau = 0$. However, we do not know if any non-constant function of τ is a lower bound on rendezvous cost in some graphs. (Recall that the cost of our algorithm contains a factor $\sqrt{\tau}$.) Hence the following problem remains open:

Does there exist a deterministic rendezvous algorithm whose cost is polynomial in n and l (or even in n and $\log l$) but independent of τ?

References

1. R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovász, and C. Rackoff, Random walks, universal traversal sequences, and the complexity of maze problems, Proc. 20th Annual Symposium on Foundations of Computer Science (FOCS'1979), 218-223.
2. S. Alpern, The rendezvous search problem, SIAM J. on Control and Optimization 33 (1995), 673-683.
3. S. Alpern, Rendezvous search on labelled networks, Naval Reaserch Logistics 49 (2002), 256-274.
4. S. Alpern and S. Gal, The theory of search games and rendezvous. Int. Series in Operations research and Management Science, Kluwer Academic Publisher, 2002.
5. J. Alpern, V. Baston, and S. Essegaier, Rendezvous search on a graph, Journal of Applied Probability 36 (1999), 223-231.
6. S. Alpern and S. Gal, Rendezvous search on the line with distinguishable players, SIAM J. on Control and Optimization 33 (1995), 1270-1276.
7. E. Anderson and R. Weber, The rendezvous problem on discrete locations, Journal of Applied Probability 28 (1990), 839-851.
8. E. Anderson and S. Essegaier, Rendezvous search on the line with indistinguishable players, SIAM J. on Control and Optimization 33 (1995), 1637-1642.
9. E. Anderson and S. Fekete, Asymmetric rendezvous on the plane, Proc. 14th Annual ACM Symp. on Computational Geometry, 1998.
10. E. Anderson and S. Fekete, Two-dimensional rendezvous search, Operations Research 49 (2001), 107-118.
11. 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 (1998), 1880-1889.
12. V. Baston and S. Gal, Rendezvous search when marks are left at the starting points, Naval Res. Log. 48 (2001), 722-731.
13. S.A. Cook and P. McKenzie, Problems complete for deterministic logarithmic space, Journal of Algorithms 8 (5) (1987), 385-394.
14. D. Coppersmith,, P. Doyle, P. Raghavan, and M. Snir, Random walks on weighted graphs, and applications to on-line algorithms, Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC'1990), 369-378.
15. D. Coppersmith, P. Tetali, and P. Winkler, Collisions among random walks on a graph, SIAM J. on Discrete Math. 6 (1993), 363-374.
16. A. Dessmark, P. Fraigniaud, and A. Pelc, Deterministic rendezvous in graphs, Proc. 11th European Symposium on Algorithms (ESA'2003), LNCS 2832, 184-195.
17. M. Dyer, A. Frieze, and R. Kannan, A random polynomial time algorithm for estimating volumes of convex bodies, Proc. 21st Annual ACM Symposium on Theory of Computing (STOC'1989), 375-381.

18. 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, 247-258.
19. S. Gal, Rendezvous search on the line, Operations Research 47 (1999), 974-976.
20. A. Israeli and M. Jalfon, Token management schemes and random walks yield self stabilizing mutual exclusion, Proc. 9th Annual ACM Symposium on Principles of Distributed Computing (PODC'1990), 119-131.
21. E. Kranakis, D. Krizanc, N. Santoro and C. Sawchuk, Mobile agent rendezvous in a ring, Proc. 23rd International Conference on Distributed Computing Systems (ICDCS'2003), 592-599.
22. W. Lim and S. Alpern, Minimax rendezvous on the line, SIAM J. on Control and Optimization 34 (1996), 1650-1665.
23. T. Schelling, The strategy of conflict, Oxford University Press, Oxford, 1960.
24. L. Thomas, Finding your kids when they are lost, Journal on Operational Res. Soc. 43 (1992), 637-639.
25. X. Yu and M. Yung, Agent rendezvous: a dynamic symmetry-breaking problem, Proc. International Colloquium on Automata, Languages, and Programming (ICALP'1996), LNCS 1099, 610-621.