Chapter 1
Deterministic Symmetric Rendezvous in Arbitrary Graphs
Overcoming Anonymity, Failures and Uncertainty

Jérémie Chalopin, Shantanu Das, and Peter Widmayer

Abstract We consider the rendezvous problem of gathering two or more identical agents that are initially scattered among the nodes of an unknown graph. We discuss some of the recent results for this problem focusing only on deterministic algorithms for the general case when the graph topology is unknown, the nodes of the graph may not be uniquely labeled and the agents may not be synchronized with each other. In this scenario, the objective is to solve rendezvous whenever deterministically feasible, while optimizing on the amount of movement by the agents or the memory required (for the nodes or the agents) in the worst case. Further we also investigate some special scenarios such as (i) when the graph contains some *dangerous* nodes or, (ii) when there is no consistent ordering on the edges of a node. We present positive results, complexity analysis and some general techniques for dealing with such worst case scenarios for the symmetric rendezvous problem.

1.1 Introduction

The problem of rendezvous requires two or more entities (called *agents*) located in distinct vertices of a graph, to meet at one vertex of the graph. This problem occurs in many natural contexts [3] and requires different strategies depending on the scenario and the particular objective. In the original definition of the problem [2], the objective was to minimize the expected time

Jérémie Chalopin
LIF, CNRS & Aix-Marseille University, France.
e-mail: jeremie.chalopin@lif.univ-mrs.fr

Shantanu Das
BGU & Technion-Israel Institute of Technology, Israel.
e-mail: shantanu@tx.technion.ac.il

Peter Widmayer
Institute of Theoretical Computer Science, ETH Zürich, Switzerland.
e-mail: widmayer@inf.ethz.ch

to meet. If we are restricted to deterministic strategies, the objective may be to minimize the worst-case time to meet, over all possible starting configurations. Moreover if there is no common notion of time, we may wish to minimize the total distance traveled by the agents until rendezvous. In some cases, there are other parameters to consider, for example the size of memory used by the agents or the number of additional resources (e.g. flags for marking) used by the agents.

This chapter considers the deterministic rendezvous of two or more agents in a finite connected graph, placed initially at locations chosen by an adversary. The agents are assumed to be identical and they execute the same algorithm, without global knowledge about the graph. Other than finiteness and connectivity, we make no other assumptions about the topology of the graph. In an arbitrary connected graph, it is not always possible to solve rendezvous using deterministic means. For instance consider a ring of size n, where two agents are placed at a distance of $n/2$ from each other; if each agent follows the same strategy (any combination of moving clockwise, counterclockwise or remaining stationary) the agents may forever be the same distance apart from each-other. In most cases, the ability to distinguish vertices in some way allows the agents to rendezvous even if they are using identical strategies. Given any graph and the starting locations of the agents in the graph, it is possible to determine whether rendezvous is possible for the particular instance. Thus, it is possible to characterize the instances where deterministic rendezvous is feasible. The prior knowledge of certain graph parameters (such as the size or diameter) or the ability to mark vertices of the graph also influences the feasibility of rendezvous.

The model considered here is very generic in the sense that we do not assume any global clock (the agents act asynchronously), nor do we assume unique identifiers for the nodes of the graph or for the agents (the graph and the agents may be anonymous); and the graph topology is not known to the agents (i.e. the topology could be any arbitrary connected graph). In stronger models, e.g. when the agents have distinct identifiers [12, 17] or, when they are synchronous [13, 14], or if the environment is restricted to specific topologies such as the ring [20, 23], grid [4] or tree [14] topologies, then it becomes easier to solve rendezvous and the set of solvable instances may become relatively larger. For results on rendezvous in such models, please see the recent survey [24]. Another significant difference between the results in this paper and those of [24] is that we allow the agents to meet only at a node, whereas most results from the above paper also allow meeting on an edge when two agents are traversing it from opposite sides[1]. The rendezvous problem has also been studied in a completely different model where the agents move in a continuous terrain [19] or in a graph [22] but have global visibility. Finally there exist many results on solving rendezvous using randomized algorithms (see [3] for a survey).

[1] This difference implies that in our model it is not possible to rendezvous even on the trivial graph consisting of two nodes and a single edge.

This chapter is organized as follows. The next section defines the model and the problem and describes some basic properties of graphs which we use in solving rendezvous. Section 1.3 provides a characterization of those instances where deterministic rendezvous is possible. In the rest of the chapter, we focus on the problem of Rendezvous-with-Detect where agents solve rendezvous whenever possible and otherwise detect the fact that rendezvous is not possible. Section 1.4 provides some minimum conditions required for solving the problem. We present algorithms for solving Rendezvous-with-Detect both for the model where agents are not allowed to mark nodes (Section 1.5) and the model where marking is allowed (Section 1.6). In section 1.7, we consider the model where each agent is provided with a pebble, allowing it to mark at most one node of the graph. We also discuss how to tolerate failures and uncertainties in this model. Finally, in section 1.8, we study rendezvous in dangerous environments where some of the agents may disappear (e.g. they are devoured by some faulty node), and show how to rendezvous the surviving agents. Section 1.9 concludes this chapter with a discussion of some open directions.

1.2 The Model and Basic Properties

The Environment. The environment is represented by a simple undirected connected graph $G = (V(G), E(G))$ and a set $\mathcal{Q}$ of mobile agents that are located in the nodes of G. The initial placement of the agents is denoted by the function $p : \mathcal{Q} \to V(G)$. We denote such a distributed mobile environment by $(G, \mathcal{Q}, p)$ or by (G, χ_p) where χ_p is a vertex-labeling of G such that $\chi_p(v) = 1$ if there exists an agent a such that $p(a) = v$, and $\chi_p(v) = 0$ otherwise. For simplicity, we assume the agents to be initially located in distinct nodes, but the algorithms can be generalized to the case when two or more agents start from the same location (e.g. if two agents happen to be initially co-located, they will move together as a single merged agent).

For the rest of this paper, $n = |V(G)|$ and $m = |E(G)|$ denotes the numbers of vertices and of edges of G, while $k = |\mathcal{Q}|$ denotes the number of agents. We shall use the words vertex and node interchangeably.

In order to enable navigation of the agents in the graph, at each node $v \in V(G)$, the edges incident to v are distinguishable to any agent a at node v. In other words, there is a bijective function

$$\delta_{a,v} : \{(v, u) \in E(G) : u \in V(G)\} \to \{1, 2, \ldots d(v)\}$$

which assigns unique labels to the edges incident at node v (where $d(v)$ is the degree of v). The function $\delta_a = \{\delta_{a,v} : v \in V(G)\}$ is called the local orientation or port-numbering [2] and it is usually assumed that all agents have

[2] The labels on the edges may correspond to port numbers on a network

the same consistent port-numbering (i.e. $\delta_a = \delta, \forall a \in \mathcal{Q}$). In Section 1.7.2, we shall consider the special case when this is not true. For the rest of the paper, we assume a common port-numbering δ (and thus remove the subscript a).

The vertices of G are labeled over the set of symbols L by $\lambda\colon V(G) \to L$ which is the labeling function. However, note that this labeling is not necessarily injective, i.e. two vertices may have the same label. This means that we must design algorithms that work for any such labeling, and in particular for the constant labeling which labels all nodes with the same label $c \in L$ (in this case, the nodes of the graph are said to be *anonymous*).

The environment is thus represented by the tuple $(G, \lambda, \mathcal{Q}, p, \delta)$ or equivalently by $(G, \lambda, \chi_p, \delta)$. In case the nodes of the graph are anonymous, we shall omit λ.

The Agents. Each agent a starts from the node $p(a)$, called the *homebase* of a, and executes a sequence of steps. The agents start from the same initial state but may not necessarily start at the same time, and every action they perform (computing, moving, etc.) takes a finite but otherwise unpredictable amount of time (i.e. the actions of the agents are not synchronized). The actions that an agent a located at a node v can perform depend on the state of the agent and the state of the node v (including the degree of v, the label of v, and the presence of other agents or marks left by other agents). An agent can see another agent only when they are both located at the same node. However, an agent may not even detect the presence of another agent if both are traversing the same edge. Two agents may traverse the same edge at different speeds; thus if agents a and b start traversing the same edge (u, v) one after the other, it is possible that agent b arrives at the other end-point earlier than agent a.

Communication model: Whiteboards and Tokens. As mentioned before, two agents may communicate (i.e. exchange information) directly only when they are at the same node. To facilitate the task of rendezvous, sometimes the agents may be allowed to leave marks on a node as a signal for other agents. In the *whiteboard* model, the agents communicate by reading and writing information on public whiteboards locally available at the nodes of the network. Each node $v \in G$ has a whiteboard (which is a shared region of its memory) and any agent visiting node v can read or write to the whiteboard. Access to the whiteboard is restricted by fair mutual exclusion, so that at most one agent can access the whiteboard of a node at the same time, and any requesting agent will be granted access within finite time.

A more restrictive model is the *token* model, where no whiteboards are available but each agent has one or more identical *tokens* (sometimes called pebbles) to mark nodes. An agent that contains a token can place it on a node v before leaving the node; this token will be visible to any agent visiting node v, i.e. the visiting agent can determine whether or not there is a token at that node. Similar to the whiteboard model, we assume mutually exclusive access to node; Thus two agents may not place their tokens at the same node simultaneously, they must do so sequentially. The tokens are moveable, i.e.

an agent can pick up a token, carry the token and place it on a different node that the agent visits.

Cost Measures. The cost of an algorithm can be the time taken until rendezvous is achieved. Since we consider the actions of the agents to be asynchronous, a more useful measure for the efficiency of the algorithm is the amount of movement made by the agents, called the move complexity. In other words, whenever an agent traverses an edge of the graph, this incurs a unit cost and the total cost of the algorithm is the total number of edge-traversals made by all the agents together, in the worst case execution of the algorithm.

Other than optimizing on the time or the movement cost, one can consider the space cost of an algorithm e.g. the memory that needs to be allocated at each node of the graph, or the memory used by an agent during the algorithm. Thus, we associate three different cost measures for a rendezvous algorithm: (i) Movement Cost, (ii) Node Memory, and (iii) Agent Memory.

Problem Definition

Definition 1 (Rendezvous). Given a distributed mobile environment (G, λ, $\mathcal{Q}$, p, δ), an algorithm $\mathcal{A}$ is said to solve *rendezvous* if for any distributed execution of the algorithm by the agents, there exists a node $v \in G$ such that all agents in $\mathcal{Q}$ eventually reach node v and do not move thereafter.

In the definition above, we do not require the agents to terminate explicitly (i.e. an agent may not be aware when rendezvous has been achieved). Note that even though we consider only deterministic algorithms, the outcome of the algorithm may depend on the particular sequence of events and actions during the (asynchronous) execution of the algorithm by the individual agents. We define a *distributed execution* of an algorithm as one possible sequence of actions and events that is consistent with the environment and the algorithm.

We say that rendezvous is feasible in $(G, \lambda, \mathcal{Q}, p, \delta)$, if and only if there exists a deterministic algorithm that solves *rendezvous* in $(G, \lambda, \mathcal{Q}, p, \delta)$.

Definition 2 (Rendezvous-with-Detect). Given a distributed mobile environment $(G, \lambda, \mathcal{Q}, p, \delta)$, an algorithm is said to solve *Rendezvous-with-Detect* if the following holds for any distributed execution of the algorithm. If rendezvous is feasible in $(G, \lambda, \mathcal{Q}, p, \delta)$, then all agents in $\mathcal{Q}$ must eventually terminate at one unique node v of G and if not, then each agent must terminate in its homebase and output "Rendezvous is not solvable".

When there are more than 2 agents, i.e. $|\mathcal{Q}| > 2$, we can define the concept of *partial rendezvous* where at least $w < |\mathcal{Q}|$ agents are required to gather at a node of the graph. This will be discussed further in section 1.8.

Properties of Graphs: Coverings and Universal Exploration Sequences

The notions presented in this section were introduced in [7]. Any connected (undirected) graph G can be represented by a strongly connected symmetric digraph $D = Dir(G)$, where each edge of G is represented by a pair of symmetric arcs in D, one in each direction. In this section, we present some definitions and results related to directed graphs and their coverings, which we use to characterize the solvable instances for rendezvous. A directed graph (digraph) $D = (V(D), A(D), s_D, t_D)$ possibly having parallel arcs and self-loops, is defined by a set $V(D)$ of vertices, a set $A(D)$ of arcs and by two maps s_D and t_D that assign to each arc two elements of $V(D)$: a source and a target (in general, the subscripts will be omitted). A *symmetric* digraph D is a digraph endowed with a symmetry, that is, an involution $Sym : A(D) \to A(D)$ such that for every $a \in A(D), s(a) = t(Sym(a))$ and $Sym(Sym(a)) = a$. Given a simple connected graph G, a vertex labeling function λ, and a port-numbering δ, we will denote by $(Dir(G), \lambda, \delta)$ the labeled digraph constructed in the following way. The vertices of $Dir(G)$ are the vertices of G and they have the same labels as in (G, λ). Each edge $\{u, v\}$ is replaced by two arcs $a_{(u,v)}, a_{(v,u)} \in A(Dir(G))$ such that $s(a_{(u,v)}) = t(a_{(v,u)}) = u$, $t(a_{(u,v)}) = s(a_{(v,u)}) = v$, $\delta(a_{(u,v)}) = (\delta_u(v), \delta_v(u))$, $\delta(a_{(v,u)}) = (\delta_v(u), \delta_u(v))$ and $Sym(a_{(u,v)}) = a_{(v,u)}$.

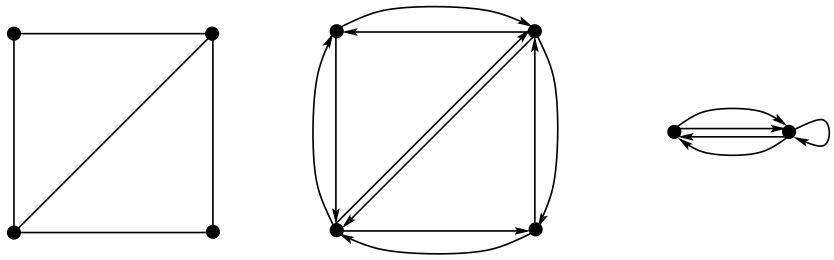

Fig. 1.1 A graph G, the corresponding digraph Dir(G), and its minimum-base H

A *covering projection* is a homomorphism φ from D to D' satisfying the following: (i) For each arc a' of $A(D')$ and for each vertex v of $V(D)$ such that $\varphi(v) = v' = t(a')$ there exists a unique arc a in $A(D)$ such that $t(a) = v$ and $\varphi(a) = a'$. (ii) For each arc a' of $A(D')$ and for each vertex v of $V(D)$ such that $\varphi(v) = v' = s(a')$ there exists a unique arc a in $A(D)$ such that $s(a) = v$ and $\varphi(a) = a'$. A symmetric digraph D is a *symmetric covering* of a symmetric digraph D' via a homomorphism φ if φ is a covering projection from D to D' such that for each arc $a \in A(D)$, $\varphi(Sym(a)) = Sym(\varphi(a))$.

A digraph D is *symmetric-covering-minimal* if there does not exist any graph D' not isomorphic to D such that D is a symmetric covering of D'.

The notions of coverings extend to labeled digraphs in an obvious way: the homomorphisms must preserve the labeling. Given a simple labeled graph (G, λ) with a port-numbering δ, we say that (G, λ, δ) is symmetric-covering-minimal if $(Dir(G), \lambda, \delta)$ is symmetric-covering-minimal. For any simple labeled graph (G, λ) with a port-numbering δ, there exists a unique digraph (D, μ_D) such that (i) $(Dir(G), \lambda, \delta)$ is a symmetric covering of (D, μ_D) and (ii) (D, μ_D) is symmetric-covering-minimal. This labeled digraph (D, μ_D) is called the *minimum base* of (G, λ, δ).

The main result that we will use from the theory of graph coverings is the following. Given an environment $(G, \lambda, \chi_p, \delta)$, if the corresponding labeled digraph $(Dir(G), \mu_G)$ is not symmetric-covering-minimal, i.e. $(Dir(G), \mu_G)$ covers a smaller digraph (H, μ_H), then the vertices of G can be partitioned into equivalence classes, each of size $q = |V(G)|/|V(H)|$ such that the vertices in the same class are symmetric and indistinguishable from each other. This is also related to the concept of views introduced in [25]. Nodes having the same view belong to the same equivalence class.

Definition 3. Given a labeled graph (G, λ) with a port numbering δ, the *view* of a node v is the infinite rooted tree denoted by $T_G(v)$ defined as follows. The root of $T_G(v)$ represents the node v and for each neighbor u_i of v, there is a vertex x_i in $T_G(v)$ (labeled by $\lambda(u_i)$) and an edge from the root to x_i with the same labels as the edge from v to u_i in (G, δ). The subtree of $T_G(v)$ rooted at x_i is again the view $T_G(u_i)$ of node u_i.

For traversal of an unknown graph we will use the notion of a *Universal Exploration Sequence* (UXS) [21]. For any node $u \in G$, we define the ith successor of u, denoted by $succ(u, i)$ as the node v reached by taking port number i from node u (where $0 \leq i < d(u)$). Let $(a_1, a_2, \ldots, a_k)$ be a sequence of integers. An *application* of this sequence to a graph G at node u is the sequence of nodes $(u_0, \ldots, u_{k+1})$ obtained as follows: $u_0 = u$, $u_1 = succ(u_0, 0)$; for any $1 \leq i \leq k$, $u_{i+1} = succ(u_i, (p + a_i) \mod d(u_i))$, where p is the port-number at u_i corresponding to the edge $\{u_{i-1}, u_i\}$. A sequence $(a_1, a_2, \ldots, a_k)$ whose application to a graph G at any node u contains all nodes of this graph is called a UXS for this graph. A UXS for a class of graphs is a UXS for all graphs in this class. For any positive integers n, d, $d < n$, there exists a UXS of length $O(n^3 d^2 \log n)$ for the family of all graphs with at most n nodes and maximum degree at most d [1].

1.3 Feasibility of deterministic rendezvous

Deterministic rendezvous is not always possible in arbitrary graphs, as we have seen before (recall the example of the two agents symmetrically placed in a ring). Given an environment $(G, \lambda, \mathcal{Q}, p, \delta)$, the feasibility of rendezvous may depend on the structure of G, the labeling λ, the port numbering δ as

well as the initial placement of the agents. When the agents do not have the capability of marking nodes, the feasibility of rendezvous depends on the labeled graph (G, λ, δ) and not on the starting locations. This is equivalent to the feasibility of electing a leader among the nodes of a graph, a well-studied problem for which there exists a known characterization of solvable instances. The following properties are based on the results from [6, 25, 26].

Theorem 1. *Rendezvous is solvable in (G, λ, δ) irrespective of the number of agents and their starting locations if and only if (G, λ, δ) is symmetric-covering-minimal with respect to any covering projection that preserves the edge-labeling δ and the node-labeling λ.*

On the other hand, if the agents are allowed to mark the nodes of the graph then the placement p of the agents in G influences the solvability of rendezvous. In this case, we can assume that the starting locations of the agents are distinctly labeled by the function χ_p and thus consider the node-labeling $\lambda' = \lambda \times \chi_p$.

Theorem 2. *Rendezvous is solvable in the environment $(G, \lambda, \mathcal{Q}, p, \delta)$ if and only if (G, λ', δ) is symmetric-covering-minimal with respect to any label-preserving covering projection, where $\lambda' = \lambda \times \chi_p$.*

We can assume that the edge-labeling and node labeling of the graph is given by an adversary. Thus, it makes sense to characterize the family of graphs where rendezvous is possible for any labeling (assuming that the labeling provides a local orientation at each node).

Theorem 3. *Given any connected graph G, the following statements are equivalent:*

1. *For any port-numbering δ, and any placement χ_p of agents in G, rendezvous can be solved in (G, χ_p, δ);*
2. *For any port-numbering function δ, each vertex of (G, δ) has a distinct view ;*
3. *There is no partition $V_1, V_2, \ldots V_k$ of $V(G)$ with $k \in [1, |V(G)| - 1]$ such that for any distinct $i, j \in [1, k]$, the following conditions hold:*
   - *(i) $G[V_i]$ is d-regular for some d, and if d is odd, it contains a perfect matching,*
   - *(ii) $G[V_i, V_j]$ is regular.*
4. *$Dir(G)$ is symmetric-covering-minimal.*

1.4 Impossibility Results

From the results of the previous section, we know rendezvous can be solved only in an environment $(G, \lambda, \mathcal{Q}, p, \delta)$ where the corresponding labelled graph

(G, λ', δ) is symmetric-covering minimal. Given such an instance, it is possible to construct another instance $(H, \lambda'_H, \delta_H)$ such that $|V(H)| = 2|V(G)|$ and $(H, \lambda'_H, \delta_H)$ covers (G, λ', δ), and thus, rendezvous is not possible in $(H, \lambda'_H, \delta_H)$. Any algorithm that solves *Rendezvous-with-Detect* must be able to distinguish between these two instances. It is not possible to distinguish between these two instances unless the agents are provided with some prior knowledge which allows them to deduce the size of the graph with some accuracy. In fact, if the agents know an upper bound B such that $B < 2n$ this is already sufficient to solve *Rendezvous-with-Detect*. (Recall that for any graph H that covers G and is not isomorphic to G, the size of H must be strictly a multiple of the size of G and thus $|V(H)|$ is at least twice of $|V(G)|$.)

Theorem 4. *The knowledge of only an arbitrary upper bound on n is not sufficient for solving Rendezvous-with-Detect in an environment $(G, \lambda, \mathcal{Q}, p, \delta)$.*

We now consider the move complexity of Rendezvous-with-Detect. It is easy to see that each edge of the graph must be traversed by at least one agent. Moreover, in a symmetric environment (e.g. a ring with agents placed equidistant apart) each agent may need to make $O(n)$ moves before it can detect the impossibility of rendezvous. This gives us the following lower bound.

Theorem 5. *Solving Rendezvous-with-Detect with k agents in an arbitrary graph of n nodes and m edges requires $\Omega(m + nk)$ moves in the worst case.*

1.5 Rendezvous without Marking

In this section we assume that the agents have no means of marking the nodes of the graph (i.e. no whiteboards or tokens are available). The knowledge of n (or, at least some upper bound on it) is required to even explore the graph unless the graph happens to be a tree. In asymmetric trees, rendezvous is possible without marking and without knowledge of n. It is possible to traverse an anonymous tree and find the central edge or central node in the tree (every tree has either a central node or a central edge). The usual technique for rendezvous is to gather at the central node or at one of the endpoints of the central edge. In the latter case, the agent needs to do a comparison of the subtrees at either end of the central edge e, in order to choose among the two end-points of e. This problem has been studied for agents having small memory (see Section 1.5.2).

1.5.1 Agents having Unbounded Memory

When an upper bound on n is known a priori, and the agents have sufficient memory, it is possible to solve rendezvous in an arbitrary graph (G, λ, δ)

by constructing the minimum-base of the labeled graph and then moving to a unique node of the minimum-base. Note that according to Theorem 1, rendezvous is solvable in this case only if (G, λ, δ) is covering minimal. If that condition is satisfied then the constructed minimum-base is isomorphic to G and thus all the agents will reach the same node, hence solving rendezvous.

In case the exact value of n is provided, it is possible to use the same algorithm to check for symmetric-covering-minimality and thus, solve Rendezvous-with-Detect. We now discuss the algorithm (see [9] for more details). The first part of the algorithm is a traversal of the graph visiting every vertex of G at least once. The second part is the classification of the visited vertices into equivalence classes. Initially all vertices are put in the same class and in subsequent rounds, the algorithm refines the classes until each class corresponds to one vertex of the minimum-base. For the traversal we use a UXS $U(N, d)$ where $N \geq n$ is an upper bound on n and d is some upper bound on the maximum degree of the graph G. We now describe the class refinement process.

Algorithm 1: Class-Refinement(N)

```
Let v_1, v_2, ... v_t be the sequence of nodes visited by U(N,d), possibly containing
duplicate nodes ;
Follow U(N,d) and for each node v_i do
    Store the labels of each edge incident to v_i;
Compute the number of 1-classes and store a distinguishing path for each pair of
distinct classes ;
k := 2;
repeat
    Follow U(N,d) and for each node v_i do
        for each edge (v_i, w) incident to v_i do
            Compute the (k-1)-class of w (using the distinguishing paths);
            Store the label of (v_i, w) and the index of the (k-1)-class of w ;
    Compute the number of k-classes and store a distinguishing path for each pair
    of distinct k-classes ;
    Increment k;
until the number of k-classes is equal to the number of (k-1)-classes;
Move to a vertex of class one;
```

Given a graph G and node u of G and a sequence of edge-labels $Y = ((p_1, q_1), (p_2, q_2), \ldots, (p_j, q_j))$, we say that Y is *accepted* from u if there exists a path $P = (u = u_0, u_1, \ldots, u_j)$ in G such that $\delta(P) = Y$, i.e. for each i, $1 \leq i \leq j$, $(p_i, q_i) = \delta(u_{i-1}, u_i)$. For any $k > 0$, two vertices u, v that have the same view up to depth k are said to be k-equivalent; we denote it by $u \sim_k v$. The k-class of u is the set of all vertices that are k-equivalent to u and this set is denoted by $[u]_k$. Given any two k-classes C, C', a (C, C')-*distinguishing path* is a sequence of edge-labels $Y_{C,C'} = ((p_1, q_1), (p_2, q_2), \ldots, (p_j, q_j))$ such that $Y_{C,C'}$ is accepted from each node $u \in C$ and it is not accepted from

any node $v \in C'$. Given any two distinct k-classes C, C', either there exists a (C, C')-*distinguishing path* of length at most k, or there exists a (C', C)-*distinguishing path* of length at most k.

For $k = 1$, it is easy to determine the k-class of any node v by traversing each edge incident to v and noting the labels. From this information, one can find the distinguishing paths for any pair of 1-classes. For $k \geq 2$, it is possible to identify the k-classes and the corresponding distinguishing paths (from knowledge of the $k-1$ classes) using the properties below.

Property 1. For $k \geq 2$, two nodes u and v belong to the same k-class, i.e. $[u]_k = [v]_k$, if and only if (i) $[u]_1 = [v]_1$ and (ii) for each i, $0 \leq i \leq deg_G(u) = deg_G(v)$, the ith neighbor u_i of u and the ith neighbor v_i of v belong to the same $(k-1)$-class and $\delta(u, u_i) = \delta(v, v_i) = (i, j)$, for some $j \geq 0$.

Theorem 6 ([9]). *Algorithm 1 builds the quotient graph of any graph of size $n \leq N$ in $O(|U(N,d)| \cdot n^3 d)$ moves and requires $O(n^3 \log n + |U(N,d)| \log d)$ memory for each agent.*

There exists a UXS for graphs of size at most N and maximum degree at most d, that is of length $O(N^3 d^2 \log N)$ [1]. Using such a sequence for the traversal gives us an algorithm of move complexity $O(N^3 n^3 d^3 \log N)$ for solving rendezvous.

1.5.2 Agents having Little Memory

In the algorithms discussed above, the agent needs to have enough memory to construct and to remember a map of the graph or a part of it. In this section we consider the effect of limiting the memory of the agent. The task of rendezvous in tree networks has been studied in synchronous environments for agents with small memory and it was shown that logarithmic memory is required for rendezvous even on the line [14]. Note that this lower bound does not apply directly in our setting since the set of solvable instances of rendezvous is strictly larger in a synchronous environment than in an asynchronous one when the agents cannot mark the vertices. However, it is well known that $o(\log n)$ memory is not sufficient for exploration of an arbitrary graph with termination, if marking is not allowed. Consequently, if the agents cannot mark the nodes, one can show that $\Omega(\log n)$ bits of memory are necessary to solve rendezvous of two agents in an arbitrary graph in an asynchronous environment.

In a synchronous environment without the ability to mark nodes, it is known [13] that $O(\log n)$ memory is sufficient for rendezvous of two agents starting on asymmetric positions in an arbitrary graph (even if the agents do not necessarily start at the same time). The idea of the algorithm in [13] is to obtain a unique ordering on distinct equivalence classes of nodes (without having to construct the views of the nodes). Each agent can then use the

index given to its initial position as its unique identifier and rendezvous can be achieved using the standard algorithm for agents having distinct identifiers in synchronous environments.

In the asynchronous setting, when the agent cannot mark nodes, we know that the starting positions of the agents cannot be used to break symmetry. Thus, rendezvous is solvable only if (G, λ, δ) is symmetric-covering-minimal. If each agent initially knows an upper bound on the size of the graph, the agent can execute the first part of the algorithm of [13] to distinguish between equivalence classes of nodes and to order them. Thereafter, each agent could move to a node that belongs to the class appearing first in this ordering. If (G, λ, δ) is symmetric-covering-minimal, this node is unique and the agents would have achieved rendezvous. Consequently, we have the following theorem.

Theorem 7. *Agents with $O(\log n)$ memory can solve rendezvous in any environment (G, λ, δ) where deterministic rendezvous is feasible without marking.*

1.6 Rendezvous with Marking

In this section, we assume that the agents can write on whiteboards present in the nodes of G. If we assume no bounds on the memory available to the agent or at a node then there is an optimal algorithm to solve rendezvous (or Rendezvous-with-Detect) for two agents using $\Theta(m)$ moves. For the general case of $k \geq 2$ agents, this generalizes to an algorithm that requires $O(mk)$ moves to solve Rendezvous-with-Detect and $O(m \log k)$ to solve rendezvous.

The algorithm proceeds in two phases. In the first phase, the agents construct a spanning forest of the graph using a distributed DFS-type algorithm (described below as procedure DDFS). At the end of this procedure there is exactly one agent in each tree in the forest and each agent a has a map of the tree that it belongs to (we call this the agent's territory T_a). The second phase of the algorithm is a competition between neighboring agents, during which each losing agent merges its territory with the corresponding winning agent. This process is repeated with the objective of eventually forming a single tree spanning the graph G so that all the agents gather at the root of this spanning tree. We show that this is always possible whenever the condition of Theorem 2 is satisfied.

Procedure DDFS: An agent A starts from its homebase a depth-first search traversal marking the nodes that it visits (unless they are already marked) and labeling them with numbers $1, 2, 3, \ldots$ etc. Each node marked by the agent and the edge used to reach it are added to its tree. If the agent reaches an already marked node, it backtracks to the previous node and tries the other edges incident to the node. The agent stops when there are no unexplored edges incident to the nodes of its tree. This tree is the territory T_A of the agent.

Partial-view (PV): Based on the territory of an agent, we define the *Partial-View* PV_A of an agent A having territory $\mathcal{T}_A$, as the finite rooted tree, such that: (i) The root corresponds to the homebase v_0 of agent A. (ii) For every other node v_i in $\mathcal{T}_A$, there is a vertex x_i in PV_A. (iii) For each edge (v_i, v_j) in $\mathcal{T}_A$, there is an edge (x_i, x_j) in PV_A. (iv) For each outgoing edge $e = (v_i, u_i)$ such that $v_i \in \mathcal{T}_A$ but $e \notin \mathcal{T}_A$, PV_A contains an extra vertex y_i (called an external vertex) and an edge $\hat{e} = (x_i, y_i)$ that joins x_i to it. (v) Each edge in PV_A is marked with two labels, which are same as those of the corresponding edge in G. (vi) Each vertex x_i in PV_A is labeled with $\lambda(v_i)$ and $\chi_p(v_i)$, where v_i is the node in G corresponding to x_i. (vii) Each vertex is also labeled with the numeric identifier assigned to the corresponding node during procedure DDFS.

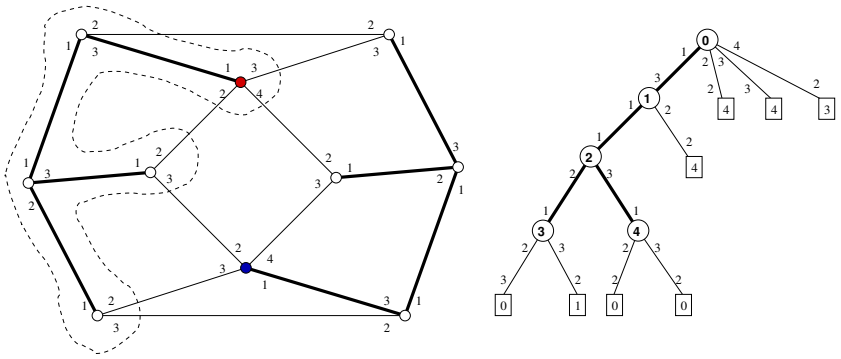

Fig. 1.2 Territories and Partial-Views: (a) A graph G with 10 nodes and two agents A and B (whose territories are marked by **bold** edges). (b) The *Partial-View* PV_A for agent A

The algorithm proceeds by comparing the partial-views of neighboring agents (we use a fixed ordering on the partial-views). We say that an agent A is a neighbor to agent B, if there exists an edge (u, v) such that $u \in T_A$, $(u, v) \notin T_A$ and $v \in T_B$. By this definition, an agent may be its own neighbor. The communication between neighbors works as follows. To send any information w, the agent writes w on each whiteboard of its territory (function "WRITE-ALL"). To read the partial-view of neighboring agents, an agent visits each external node x and reads the contents of the whiteboard at x (function "READ-PV"). In any round i, if agent A reads a partial-view $PV_{i,x}$ greater than its own partial-view $PV_{i,A}$ in this round, then agent A is defeated (i.e. it becomes passive and does not participate in the algorithm anymore) and the edge connecting node x to the tree T_A is used to merge the two trees. This process is repeated for k iterations or until the territory of an agent spans the whole graph.

The algorithm assumes the prior knowledge of $k = |\mathcal{Q}|$. Alternately if the value of n is known (but not k) then the algorithm may be modified accordingly to use this information. In this case, the main loop of the algorithm will be executed for at most n iterations and the agent will terminate the algorithm successfully if its territory contains n nodes.

Algorithm 2: Make-Tree(k)

```
Execute procedure DDFS to construct the territory T_A;
PV_{1,A} ← COMPUTE-PV(T_A) ;
for phase i = 1 to k do
    if Number of Agents in T_A is k then
        Collect all agents to root;
        Return("Success");
    WRITE-ALL(PV_{i,A}, i);
    S ← READ-PV (i);
    State ← COMPARE-PV(PV_{iA}, S);
    if State = Passive then
        SEND-MERGE(i);
        WRITE-ALL("Defeated", i);
        Return to homebase and execute WAIT();
    else
        RECEIVE-MERGE(i);
        execute UPDATE-PV() and continue;
WRITE-ALL("Failure");
```

Lemma 1 ([15]). *Algorithm* Make-Tree *solves Rendezvous-with-Detect using $O(mk)$ moves in total and requires $O(m \log n)$ whiteboard memory for each node.*

A modified version of the algorithm solves rendezvous for all solvable instances in at most $O(m \log k)$ moves. The only modification is during the comparison of the partial-views; if the agent A finds that all neighboring agents have the same Partial-view, then agent A returns to its homebase and waits (instead of continuing with the competition rounds for k iterations). This algorithm would not have an explicit termination.

Another possible modification to the algorithm is to reduce the memory required for the whiteboards (see [15]). If the whiteboards at each node are limited to $O(\log n)$ bits, then a modified version of Algorithm *Make-Tree* can be used to solve Rendezvous-with-Detect using $O(\log n)$ bit whiteboards and $O(m^2 k)$ moves in total. The idea of this algorithm is to perform the comparisons of partial-views by traversing the territories of the neighbors. Thus the agents have to perform additional moves, but the only information that we need to write on the whiteboards is the label assigned to the node and a link to its parent in the tree.

1.7 Rendezvous using Tokens

In this section we consider the rendezvous problem in the token model where each agent has a token which they can place on any node they visit. Note that tokens used by all agents are identical (and thus indistinguishable). As opposed to the previous section, there are no public whiteboards on the nodes. If every agent puts its token on its starting location, we have a bicoloring on the nodes of the graph representing the function χ_p on $V(G)$. The agent can now execute Algorithm 1 with the following modification. The initial classification partitions the nodes into two classes–those that contain a token and those that do not! The algorithm will succeed in solving rendezvous whenever the conditions of Theorem 2 are satisfied. Moreover, the algorithm can solve Rendezvous-with-Detect, if the exact value of n is known. However this algorithm is not efficient in terms of the moves complexity as we have seen. We present below a different algorithm which is more efficient [9].

1.7.1 Rendezvous with a single unmovable token

The algorithm for rendezvous presented in this section is for two agents, though the same idea may be used to rendezvous any $k \geq 2$ agents using a more involved algorithm. We assume that an agent always places the token at its starting location. First, suppose there is a single agent exploring a graph G. The fact that the starting node r of the agent is marked and can be distinguished from other nodes, makes it easier to perform an exploration of G. The agent can perform a breadth-first traversal building a BFS-tree T rooted at r. During the traversal, whenever the agent explores a new edge and reaches a node v, it checks whether v is same as some node u in its tree. This can be done by successively applying the label-sequences for the back-paths from each node $u \in T$ to the root r, and checking if one of these hits the marked node. Based on this idea, we have an algorithm for building a map of G with a single agent starting from a unique marked homebase in G (see Algorithm 3). The algorithm maintain a BFS-tree T containing the visited nodes and a data structure called ROOT_PATHS that stores the edge-labeled path P in T from any node v to the homebase r. For such a stored path P, Start(P) refers to the node v. For any path $P = (u_0, u_1, \ldots u_t)$ in the tree T, the label sequence of path P is $\Lambda(P) = (\delta(u_0, u_1), \ldots \delta(u_{k-1}, u_t))$. Other than the tree T, the algorithm also maintains the cross-edges which together with T, give the complete map of G.

When two identical agents execute the algorithm 3 from marked homebases, it is clear that the agents will not have a map of the complete graph. However, the following properties are satisfied.

Lemma 2 ([9]). *During algorithm BFS-Tree-Construction: (i) The graph T constructed by each agent will be an acyclic connected subgraph of G, and (ii)*

Algorithm 3: BFS-Tree-Construction

```
Map := T := {r} ;
Add r to Queue;
ROOT_PATHS := {φ};
while Queue is not empty do
    Get next node v from Queue and go to v using Map;
    while node v has unexplored edges do
        Traverse the next unexplored edge e = (v, u);
        for each path P ∈ ROOT_PATHS do
            Apply sequence Λ(P) at node u ;
            if successfully reached a marked node then
                Add to Map a cross-edge from v to Start(P);
                Update the number of explored edges at the node Start(P);
                Return to node v using T and exit Loop;
            else
                Backtrack to node u ;
        if All path sequences failed to reach a marked node then
            Add a new node u to T and Map ;
            Add edge (v, u) to T and Map ;
            Insert u to Queue ;
            ROOT_PATHS := ROOT_PATHS ∪ Path_T(u, r) ;
            Backtrack to node v ;
```

if the maps constructed by the two agents are identical then the views from the two homebases are identical.

The tree constructed by an agent in the above algorithm, is similar to the territory of an agent as in Section 1.6. Due to the above properties, we know that when the maps obtained by the two agents are identical, then rendezvous is not solvable deterministically. So, we only need to consider the case when the maps are distinct. In this case if we could compare the maps of the agents, we can elect one of the agents and the agents could rendezvous at the homebase of the elected agent. This algorithm (called Algorithm RDVwithToken) was presented in [9] and we have the following result.

Theorem 8 ([9]). *Algorithm RDVwithToken solves Rendezvous-with-Detect for two agents on a graph of size n and maximum degree d, and requires $O(n^4d^2)$ moves by each agent. Each agent requires a private memory of size $O(nd \log n)$.*

1.7.2 Tolerating Failures and Uncertainty

In this section we consider two special cases. The first scenario is when tokens placed by the agent are subject to failures (i.e. they may disappear during the

execution). This problem has been studied for the ring [23] and a solution is provided for $f < k$ failures, assuming certain conditions on the parameters n and k. A more general solution for arbitrary graphs is provided in [16] which works if at least one token does not fail (irrespective of the values of n and k). The idea of the algorithm is the following. If there are no failures then any standard algorithm (e.g. the one at the beginning of this section) can be used to determine a unique rendezvous location, whenever (G, χ_p, δ) is symmetric-covering-minimal. On the other hand if there are $1 \leq f < k$ failures, then the agents whose tokens failed are distinguished from the agents whose tokens are still in their homebase. The former agents (called *Runners*) traverse the graph and carry information to each marked homebase, while the latter agents (called *Owners*) wait at their homebase to receive information from each *Runner* agent. Using this information, each *Owner* agent can determine the location of the missing tokens and thus reconstruct a map of the original environment, and solve rendezvous. The challenging part of the algorithm is to switch from the procedure for the fault-free scenario to the procedure for the faulty scenario, in case faults do occur at arbitrary times during the execution of the algorithm.

Another scenario that has been studied recently is the rendezvous of agents in graphs having no common port-numbering [8]. Note that all the algorithms considered so far are based on the fact that any two agents have the same *view* from any given vertex $v \in G$. If we consider the situation where two agents a and b may have distinct port-numbering functions δ_a and δ_b, then this assumption is no longer true. In this case, any agent a may navigate in the graph using its own port-numbering δ_a and build a map representing the minimum-base of the environment $(G, \lambda, \chi_p, \delta_a)$ but the maps built by the two agents may not necessarily be identical (though they will be isomorphic). Thus, the agents may not agree on a unique ordering of the vertices and rendezvous is not possible without any additional assumptions. Surprisingly, if the agents are provided with an additional token (i.e. each agent now has 2 identical tokens) then it is possible to solve Rendezvous-with-Detect in all environments that satisfy the conditions of Theorem 3. The algorithm that achieves this, works as follows. Once an agent a has built a map of G, the vertices of G are partitioned into automorphism classes (ignoring the port-numbering δ_a). The agents then iteratively refine this partitioning by a process of selective marking of the nodes, eventually obtaining a total order on the set of nodes. During each phase of this iterative process, an agent uses one token to mark the selected node and the other token to synchronize with other agents. The full details of the algorithm can be found in [8] and we only state the main result here.

Theorem 9 ([8]). *Two tokens per agent are necessary and sufficient to solve Rendezvous-with-Detect in the absence of common port-numbering in any environment* $(G, \lambda, \mathcal{Q}, p)$ *such that* $(Dir(G), \lambda')$ *is symmetric-covering minimal where* $\lambda' = \lambda \times \chi_p$.

1.8 Rendezvous in Dangerous graphs

We now consider the scenario where some of the nodes of the graph may be dangerous and inaccessible to the agents. This is inspired by communication networks where some nodes or links between nodes may develop faults. If an agent attempts to traverse a faulty link or to move to a faulty node it simply disappears (i.e. the agent is destroyed without leaving a trace). Given a graph with multiple faulty nodes, we can merge them all into one dangerous node x, called the *black hole*. The question of whether rendezvous can be solved in a graph containing a black hole was first studied in [18] where an algorithm was provided for ring graphs containing a single black hole. We study the problem for arbitrary graphs with both faulty nodes and faulty edges, where the agents do not have prior knowledge of the graph topology or the possible location of faults. Throughout this section we assume the whiteboard model of communication, i.e. the agents may write any information on the nodes they visit.

Since the location of a black hole is unknown and cannot be determined unless an agent falls into it and is destroyed, this means that rendezvous of all agents is not possible. Thus, the objective is to achieve rendezvous of as many agents as possible while avoiding the black hole. It can be shown that in an unknown arbitrary graph with τ links that lead to a black hole, it is not possible to solve rendezvous of more than $k-\tau$ agents [10]. Note that no agent starts from the black hole (i.e. the homebases are distinct from the black hole node). For agents starting from arbitrary locations, rendezvous of any two agents is possible only if the graph obtained after removing the black hole is connected. We now present some other conditions that must be satisfied for feasibility of deterministic rendezvous in an environment $(G, \lambda, \mathcal{Q}, p, \delta, \eta)$ where the function $\eta : E(G) \to \{0, 1\}$ denotes which edges are safe ($\eta(e) = 1$) and which are faulty ($\eta(e) = 0$). Given a graph with multiple faulty edges and faulty nodes, we can replace each faulty edge (u, v) with two edges (u, x) and (v, y) leading to two distinct (dangerous) nodes x and y respectively. We denote by τ, the number of faulty links (each faulty edge accounts for two faulty links).

We define the *extended-map* of the environment $(G, \lambda, \mathcal{Q}, p, \delta, \eta)$ as the labeled digraph (H, μ_H) such that, H consists of two disjoint vertex sets V_1 and V_2 and a set of arcs $\mathcal{A}$ as defined below:

- $V_1 = V(G)$;
- $\mu_H(v) = (\lambda(v), \chi_p(v))$, $\forall v \in V_1$;
- For every safe edge $e = (u, v) \in E(G)$, there are two arcs $a_1, a_2 \in \mathcal{A}$ such that $s(a_1) = t(a_2) = u$, $s(a_2) = t(a_1) = v$, and $\mu_H(a_1) = (\delta_u(e), \delta_v(e))$, $\mu_H(a_2) = (\delta_v(e), \delta_u(e))$.
- For every faulty edge $e = (u, v)$, there are vertices u' and $v' \in V_2$ with $\mu_H(u') = \mu_H(v') = -1$ and arcs $(u, u'), (u', u), (v, v')$ and $(v', v) \in \mathcal{A}$ with labels $(\delta_e(u), 0)$, $(0, \delta_e(u))$, $(\delta_e(v), 0)$, and $(0, \delta_e(u))$ respectively;

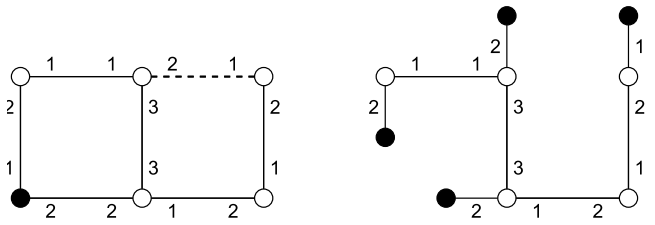

Fig. 1.3 The extended-map of an environment containing faulty edges (marked in dashed lines) and black-holes (colored black)

The *extended-map* can be thought of as a canonical representation of the environment (see Figure 1.3). It can be shown that any execution of a deterministic algorithm on the environment $(G, \lambda, \mathcal{Q}, p, \delta, \eta)$ can be simulated on the extended-map (H, μ_H). Based on this we have the following result from [10].

Theorem 10. *It is not possible to rendezvous $k-\tau$ agents in an environment whose extended-map is not symmetric-covering minimal.*

In the following we will briefly discuss an algorithm that solves the rendezvous of $k-\tau$ agents in an environment that satisfies the conditions above. We present below a lower bound on the moves complexity of any algorithm solving rendezvous of $k-\tau$ agents (see [10] for a proof).

Theorem 11. *For solving rendezvous of $(k-\tau)$ agents in an environment $(G, \lambda, \mathcal{Q}, p, \delta, \eta)$ without any knowledge other than the size of G, the agents need to make at least $\Omega(m(m+k))$ moves in total.*

We can ensure that no more than one agent dies while traversing the same link, using the *cautious walk* technique as in [18]. At each node, all the incident edges are considered to be unexplored in the beginning. Whenever an agent A at a node u has to traverse an unexplored edge $e = (u, v)$, agent A first marks link $\delta_u(e)$ as "Being Explored" and if it is able to reach the other end v successfully, it immediately returns to node u and re-marks the link $\delta_u(e)$ as "safe". During the algorithm we follow the rule that no agent ever traverses a link that is marked "Being Explored". This ensures no more than τ agents may die during the algorithm.

The algorithm is based on similar ideas as in Algorithm 2. Recall that the algorithm proceeded with each agent exploring a part of the graph and marking its territory, followed by comparison between the territories of agents over multiple rounds. In the present algorithm, several improvisations are required to account for the fact that some agents may die during the execution of the algorithm. First of all, rounds of exploration are alternated with rounds of competition between agents. During an exploration round, an agent may

fall into a black hole and die, without completing the process of marking its territory. Thus, during the competition an agent can win over another agent based on either comparison of territories or comparison of round number (a dead agent would not be able to increment its round number). When an agent A wins the territory of another agent (that may have died) there may be unexplored edges incident to this territory and the agent A needs to expand the territory during the next exploration round. Recall that it is not possible to distinguish between a dead agent and an agent that is slow in moving along an edge. This means that an agent cannot wait for another agent to complete its exploration. Thus, there could be multiple agents expanding a given territory simultaneously. Those agents which are in the same territory need to coordinate with each other in the exploration and competition tasks. This is done by communicating using messages written on the whiteboard of the root node. The algorithm ensures that there is always a unique root node in every territory during the execution of the algorithm.

At any stage of the algorithm, there are teams of agents, each team possessing a *territory* which is a connected acyclic subgraph of G (disjoint from other territories). Each team of agents tries to expand its territory until it spans a majority of the nodes. Once a team is able to acquire more than half the nodes of the network, it wins and agents from all other teams join the winning team to achieve rendezvous. We call this algorithm as Algorithm *RDV_BH*.

Theorem 12 ([10]). *Algorithm* RDV_BH *correctly solves rendezvous for $k-\tau$ agents in any network whose extended-map is symmetric-covering minimal provided the agents initially knows a bound B such that $n \leq B < 2n$. The moves complexity of algorithm* RDV_BH *is $O(m(m+k))$.*

The above result implies that the algorithm described in this section is optimal in terms of the moves complexity.

1.9 Conclusion

We considered the problem of symmetric asynchronous rendezvous in graphs whose nodes are anonymous and whose edges are locally ordered. Since the agents are identical and follow the same deterministic algorithm, solving the problem requires breaking the symmetry and finding a unique location to meet. This is possible only if there is some asymmetry in the structure of the graph (for agents that cannot mark the graph) or if the agents start from asymmetric locations within the graph (in the case when agents are allowed to mark their starting location). It is possible determine for exactly which instances rendezvous is feasible and then solve rendezvous in those cases. We presented solutions for rendezvous with detection and also discussed techniques for dealing with exceptional situations involving faulty nodes, token

failures and inconsistencies in local labelling of the edges. While all solutions studied here assume that the edges of the graph are bidirectional, some of the techniques could be extended to work for (strongly connected) directed graphs (e.g. see [11]). In general, solving rendezvous in directed graphs is more difficult due to the inability of agents to backtrack and this is one of directions for future research. Another open problem is solving rendezvous in dangerous graphs assuming the weaker model of communication when there are no whiteboards and the agents are only provided with a few pebbles.

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. In Proc. of 20th Annual Symposium on Foundations of Computer Science (FOCS), pp. 218–223, 1979.
2. S. Alpern. Hide and seek games. Seminar, Institut fur Hohere Studien, Wien, July 1976.
3. S. Alpern and S. Gal. *The Theory of Search Games and Rendezvous*. Kluwer, 2003.
4. E. Bampas, J. Czyzowicz, L. Gasieniec, D. Ilcinkas, A. Labourel. Almost Optimal Asynchronous Rendezvous in Infinite Multidimensional Grids. In Proc. 24th International Symposium on Distributed Computing (DISC), pp. 297-311, 2010.
5. V. Baston and S. Gal. Rendezvous search when marks are left at the starting points. Naval Research Logistics, 48(8):722–731, 2001.
6. P. Boldi, B. Codenotti, P. Gemmell, S. Shammah, J. Simon, and S. Vigna. Symmetry breaking in anonymous networks: Characterizations. In Proc. 4th Israeli Symposium on Theory of Computing and Systems, pp. 16–26, 1996.
7. P. Boldi and S. Vigna. Fibrations of graphs. Discrete Mathematics, 243(1-3):21–66, 2002.
8. J. Chalopin and S. Das. Rendezvous of Mobile Agents without Agreement on Local Orientation. In Proc. 37th Int. Coll. on Automata, Languages and Programming (ICALP), pp. 515-526, 2010.
9. J. Chalopin, S. Das, and A. Kosowski. Constructing a Map of an Anonymous Graph: Applications of Universal Sequences. In Proc. 14th International Conference on Principles of Distributed Systems (OPODIS), pp. 119-134, 2010.
10. J. Chalopin, S. Das, and N. Santoro. Rendezvous of Mobile Agents in Unknown Graphs with Faulty Links. In Proc. 21st International Symposium on Distributed Computing (DISC), pp. 108–122, 2007.
11. J. Chalopin, S. Das, and P. Widmayer. Rendezvous of Mobile Agents in Directed Graphs. In Proc. 24th International Symposium on Distributed Computing (DISC), pp. 282-296, 2010.
12. J. Czyzowicz, A. Labourel, and A. Pelc, How to meet asynchronously (almost) everywhere, In Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 22-30, 2010.
13. J. Czyzowicz, A. Kosowski, A. Pelc. How to meet when you forget: log-space rendezvous in arbitrary graphs. Distributed Computing 25(2): 165-178, 2012.
14. J. Czyzowicz, A. Kosowski, A. Pelc, Time vs. space trade-offs for rendezvous in trees, In Proc. 24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 1-10, 2012.
15. S. Das, P. Flocchini, A. Nayak, and N. Santoro. Effective elections for anonymous mobile agents. In Proc. 17th Symp. on Algorithms and Computation (ISAAC), pp. 732-743, 2006.

16. S. Das, M. Mihalak, R. Sramek, E. Vicari, P. Widmayer, Rendezvous of mobile agents when tokens fail anytime, In Proc. 12th Int. Conf. on Principles of Distributed Systems (OPODIS), pp. 463-480, 2008.
17. A. Dessmark, P. Fraigniaud, D. Kowalski, A. Pelc. Deterministic rendezvous in graphs, Algorithmica, 46: 69-96, 2006.
18. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Multiple agents rendezvous in a ring in spite of a black hole. In Proc. 7th Int. Conf. on Principles of Distributed Systems (OPODIS), pp. 34–46, 2003.
19. Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, Peter Widmayer: Gathering of asynchronous robots with limited visibility. Theor. Comput. Sci. (TCS) 337(1-3):147-168, 2005.
20. L. Gasieniec, E. Kranakis, D. Krizanc, X. Zhang. Optimal Memory Rendezvous of Anonymous Mobile Agents in a Unidirectional Ring. In Proc. 32nd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), pp. 282-292, 2006.
21. M. Kouckỳ, Universal traversal sequences with backtracking, J. Comput. Syst. Sci. 65:717-726, 2002.
22. R. Klasing, E. Markou, and A. Pelc, Gathering asynchronous oblivious mobile robots in a ring, Theor. Comput. Sci. 390(1):27-39, 2008.
23. E. Kranakis, D. Krizanc, and E. Markou, The mobile agent rendezvous problem in the ring, Morgan and Claypool Publishers, 2010.
24. A. Pelc. Deterministic rendezvous in networks: A comprehensive survey, Networks, 59: 331-347, 2012.
25. M. Yamashita and T. Kameda. Computing on anonymous networks: Part I–Characterizing the solvable cases. IEEE Transactions on Parallel and Distributed Systems, 7(1):69–89, 1996.
26. M. Yamashita and T. Kameda. Leader election problem on networks in which processor identity numbers are not distinct. IEEE Transactions on parallel and distributed systems, 10(9):878–887, 1999.