Asymptotically Optimal Deterministic Rendezvous∗

Fabienne Carrier
VERIMAG UMR 5104, Université Joseph Fourier
Grenoble I, France
fabienne.carrier@imag.fr

Stéphane Devismes
VERIMAG UMR 5104, Université Joseph Fourier
Grenoble I, France
stephane.devismes@imag.fr

Franck Petit
LIP6 UMR 7606, Université Pierre et Marie Curie
Paris VI, France
franck.petit@lip6.fr

Yvan Rivierre
VERIMAG UMR 5104, Université Joseph Fourier
Grenoble I, France
yvan.rivierre@imag.fr

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

In this paper, we address the deterministic rendezvous in graphs where $k$ mobile agents, disseminated at different times and different nodes, have to meet in finite time at the same node. The mobile agents are autonomous, oblivious, labeled, and move asynchronously. Moreover, we consider an undirected anonymous connected graph. For this problem, we exhibit some asymptotical time and space lower bounds as well as some necessary conditions. We also propose an algorithm that is asymptotically optimal in both space and round complexities.

Keywords: Autonomous Agents; Mobile Robot Networks; Optimality; Rendezvous.

1991 Mathematics Subject Classification: 68Q22, 68M15

∗This work has been supported in part by the ANR project SHAMAN. An early version of this work was published in [7].
1. Introduction

We consider mobile autonomous entities evolving into a discrete space. Such entities are often called *agents* or *robots*. The discrete space is modeled as a *simple undirected connected graph* where nodes represent locations and edges the possibility for an agent to move from a location to another.

*Exploration* [16,17], *leader election* [2,12], *agent naming* [13,22], and *rendezvous* [1,3,11] (also called *gathering*) are the core problems in mobile agents systems. Here, we focus on the rendezvous problem. There are many alternative definitions of this problem. Here, we consider the one of [23] where *k* agents, disseminated at different times and different nodes of the graph, have to meet at the same node before stopping to move forever.

1.1. Contributions

We study deterministic solutions for this problem under a weak scenario. The nodes are anonymous. The agents are oblivious (*i.e.*, they do not remember the past) and move asynchronously. They cannot directly communicate together, even if located at the same node. They do not have any *a priori* knowledge about each other (*such as the total number of agents*) and about the graph (*such as the topology, the number of nodes, . . .*).

The deterministic rendezvous is unsolvable under the above settings [11]. Hence, we make some additional assumptions. Namely, we use a *whiteboard* on each node (*i.e.*, a finite memory in which every agent can read and write), *local indices* on edges, and *labels* on agents. Our labeling is weak: we only assume the existence of a minimum label held at exactly one agent; other labels are not necessary uniquely assigned or comparable between them.

Our contribution is threefold. We first prove some asymptotical time and space complexity lower bounds to solve the problem in our model. We then propose an algorithm that is asymptotically optimal in both space and round complexities. Finally, we show that most of the assumptions we made are necessary to deterministically solve the rendezvous considering our initial scenario.

1.2. Related Works

The problem of rendezvous appears for the first time in [1]. This problem has been discussed in many contexts and models. For example, one can consider that agents evolve in a continuous two-dimensional euclidian space [8,9] (in this case, the problem is often referred to as *gathering*). In the context of agents in graphs, many papers assume that agents have permanent local memories, *e.g.*, [1,3,11,20,23]. Note that in these papers, no memory (*i.e.*, so-called *whiteboard*) is available on nodes of the graph. In [1,11,20,21], a rendezvous of only two agents is considered. In the three first papers, the system is assumed to be synchronous. In the last one, the system is asynchronous but the problem is relaxed: the rendezvous may occur in an edge. In [3,23], the problem is solved for any number of agents. In [23], a solution is provided for any anonymous graph, but the system is synchronous and the rendezvous is only guaranteed with a high probability. In [3], a
solution is proposed for any asynchronous anonymous graph, provided that the number of agents and the number of nodes are coprime. Also, an edge-labeling that gives a sense of direction is assumed.

Few papers are related to the rendezvous of oblivious agents on graphs, that is agents that have no persistent memory between two steps of execution. In [19], authors study the rendezvous into an oriented ring, the agents are oblivious but they are be able to snapshot the whole system at each step.

There are some papers that deal with graphs equipped with whiteboards on nodes. For example, a fault-tolerant rendezvous algorithm is proposed in [10], the algorithm uses whiteboards on nodes as well as persistent memory at each agent. In [5], the authors address the problem of self-stabilizing naming. They give a deterministic solution for tree and a probabilistic one for arbitrary graphs.

1.3. Roadmap

The rest of the paper is organized as follows. In the next section, we present our computational model. In Section 3, we exhibit asymptotical bounds on time and space complexity of rendezvous problem in our settings. Our asymptotically optimal rendezvous algorithm and its proof of correctness are given in Section 4. In Section 5, we show that several of our assumptions are necessary to deterministically solve the rendezvous. Section 6 is dedicated to concluding remarks and perspectives.

2. Preliminaries

In this section, we first define the distributed system considered in this paper. We then present the statement of the rendezvous problem.

2.1. Model

The distributed system we consider consists of a finite set of $k$ agents evolving into a discrete environment. The latter is described in the next paragraph, followed by the description of agents. Next, we describe how the agents perform some tasks by interacting with their environment.

Environment. The environment is represented by a simple undirected connected graph $G = (V, E)$ where $V$ is a finite set of $n > 1$ nodes $v_0 \ldots v_{n-1}$ and $E$ is a set of $m$ edges (or links). The subscripts $0 \ldots n - 1$ are used for notation purpose only. Indeed, nodes are anonymous, i.e., nodes can only differ by their degrees. An edge is a unordered pair of distinct nodes. Two nodes $v_i$ and $v_j$ are said to be neighbors if and only if there exists an edge $(v_i, v_j)$ in $E$.

Incident edges are distinguished at a node using locally ordered index. Without loss of generality, we assume that the outgoing links at a node $v$ are numbered from 0 to $\delta_v - 1$, where $\delta_v$ is the degree of the node $v$. $\Delta$ denotes the degree of $G$, i.e., $\Delta = \max_{v \in V} \delta_v$.

Each node is provided with a whiteboard of limited capacity. A whiteboard is a buffer
memory of a node where agents can read and write information when they are located at
the node. For ease of use, whiteboard content is represented as a set of variables.

Agents. Each agent is a mobile entity that can move from any node to one of its neighbors
by crossing the incident edge. Each agent is autonomous and oblivious. The former means
that it decides its actions by itself. The latter means that the agent has no memory of past
steps, nor inner state.

Agents have labels. We assume the existence of binary relation ≺ such that there exists
a unique label \( l \) — called the minimum label — satisfying: \( l \prec l' \) for every other label \( l' \).
We do not assume any consistency of the relation \( \prec \) for the other labels, that is: for every
two distinct non-minimum labels \( l' \) and \( l'' \), we can have either \( l' \prec l'' \), or \( l'' \prec l' \), or both.
Moreover, except for the minimum label, any label can be used by several agents. In the
following, we denote the label of agent \( a \) by the constant \( label_a \). We assume that for every
agent \( a \), \( label_a \) is encoded over \( \log_2 \mathcal{L}_{\text{max}} \) bits, that is, there exists at most \( \mathcal{L}_{\text{max}} \) distinct
labels.

Agents are unable to detect each other and/or to explicitly communicate together, even
if they are located at the same node.

The only inputs of an agent located at a node \( v \) are its own label, the set of local edge
indices at \( v \), the local index of the edge it comes from, and the local whiteboard content. No
other information is given to the agents. In particular, no agent \( a \) priori knows the labels of
the others. No agent \( a \) priori knows if its label is the minimum one or not. Moreover, the
number \( k \) of agents, the number \( m \) of edges and the number \( n \) of nodes are unknown by the
agents. An agent knows the local index of the edge it comes from, thanks to the primitive
\texttt{From}(). When an agent \( a \) appears in the system for the first time at a node \( v \), it traversed no
edge, in this case \texttt{From}() returns \( \perp \). Node \( v \) is then called home of \( a \).

Program and Moves. Every agent executes the same program that consists of a finite set
of actions. Each action is a guarded command:

\[
\langle \text{label} \rangle : \langle \text{guard} \rangle \rightarrow \langle \text{statement} \rangle
\]

A label is an agent-wide identifier of the guarded command. A guard is a boolean ex-
pression involving agent inputs. A statement is a sequence of variable assignments on the
whiteboard of the node where the agent is currently located and/or a move decision. For
ease of comprehension, guarded commands are mutually exclusive in our algorithms.

The capacity of the edges is not limited, i.e., several, possibly the \( k \) agents, are able to
traverse the same edge simultaneously. Moreover, each edge crossing is done in a finite yet
unbounded time. Finally, there is no guarantee that agents exit from an edge in the same
order they entered into it. In particular, the edges are not assumed to be First-In-First-Out
(FIFO).

Execution. The state of a node \( v \) is defined by its degree, its whiteboard content, and a list
containing one entry \((a, f)\) per agent currently in \( v \) such that \( a \) is the agent label and \( f \) is
the value returned by its \texttt{From}() function. The state of an edge \( \{p, q\} \) is defined by two lists:
the label list of the agents incoming to \( p \) and the label list of the agents incoming to \( q \). The configuration of the system is an instance of the states of each node and each edge.

An action of an agent is said to be enabled in some configuration if and only if its guard is true, considering the inputs of the agent.

An agent is said to be enabled in some configuration if and only if it is in an edge or if it is located in a node and at least one of its action is enabled. By extension, a node or an edge is said to be enabled in some configuration if and only if it holds at least one enabled agent.

An execution is a maximal sequence of configurations \( \gamma_0, \ldots, \gamma_i, \ldots \) such that \( \gamma_0 \) is an initial configuration (defined according to the algorithm and the graph) and for every \( i > 0 \), \( \gamma_i \) is obtained by atomically activating a non-empty subset of enabled edges and/or nodes on \( \gamma_{i-1} \). The activation of an edge or a node consists in activating an agent in. When an agent is activated in an edge, it atomically moves to its destination node. The activation of an agent at a node consists in atomically executing the statement of its enabled action.

The activations are managed according to a scheduler. A scheduler is a predicate over the executions. Here, we consider a distributed weakly fair scheduler. Distributed means that at each step the scheduler activates a non-empty subset of enabled edges and nodes. Weakly fair means that every continuously enabled agent is activated in finite time.

We consider that an agent \( a \) is neutralized between \( \gamma_{i-1} \) and \( \gamma_i \) if \( a \) was enabled in \( \gamma_{i-1} \) and not enabled in \( \gamma_i \), but was not activated between \( \gamma_{i-1} \) and \( \gamma_i \). The neutralization represents the following situation: an enabled agent \( a \) located in some node \( v \) becomes disabled because another agent, located in \( v \), is activated between \( \gamma_i \) and \( \gamma_{i+1} \) and modifies the whiteboard content; this change effectively made the guards of all actions of \( a \) false in \( \gamma_{i+1} \).

To compute the time complexity, we use the notion of round [6, 14]. This notion captures the execution rate of the slowest process in any execution. The first round of an execution \( e \) is the minimal prefix of \( e, \gamma_0 \ldots \gamma_i \), containing the activation or the neutralization of every agent that is enabled in the initial configuration. Let \( e_{\gamma_i} \) be the suffix of \( e \) starting from \( \gamma_i \) (the last configuration of the first round of \( e \)). The second round of \( e \) is the first round of \( e_{\gamma_i} \), and so on.

2.2. The rendezvous problem

The rendezvous problem consists in a finite process during which, for a given \( k \), \( k \) agents meet and stop at the same node. Initially no agent is present in the graph. Each agent is placed at any time, in any order, and on any node of the graph. Any execution of a rendezvous protocol eventually leads the system in a final configuration where the \( k \) agents are located at the same node.

3. Lower Bounds

In this section, we gives two asymptotic lower bounds: one for the round complexity (Theorem 4), the other for the space complexity (Theorem 5).
In the following, we say that an agent explores the graph if it traverses all edges of the graph at least once. (Consequently, all nodes are visited by the agent at least once.)

The three next lemmas are technical ones. The first is used to prove the two bounds. The two others are used in the proof of the space complexity bound only.

**Lemma 1.** Let \( \mathcal{A} \) be a rendezvous algorithm for general graphs. In any execution of \( \mathcal{A} \), at least one agent explores the graph.

**Proof.** Assume, for the purpose of contradiction, that there is a rendezvous algorithm \( \mathcal{A} \) where no agent explores the graph.

Consider an execution of \( \mathcal{A} \) for a team of agents \( a_1 \ldots a_k \) in a graph \( G_a = (V_a, E_a) \). Assume that \( a_1 \) starts first. Agents \( a_2 \ldots a_k \) can stay not activated during an arbitrary long time. By assumption, there exists an edge \( \{u_a, v_a\} \) that \( a_i \) never traverses. Let \( i \) (resp. \( j \)) the local index of this edge at \( u_a \) (resp. \( v_a \)). Let \( \mathcal{E}_{a_i} \) be the maximal prefix of execution where \( a_i \) is the only activated agent. Two cases are possible: either \( a_i \) eventually stops at some node in \( \mathcal{E}_{a_i} \) or \( \mathcal{E}_{a_i} \) can be infinitely extended.

Consider the same reasoning with a graph \( G_b = (V_b, E_b) \) and a team of agents \( b_1 \ldots b_j \). Then, there exists an edge \( \{u_b, v_b\} \) that \( b_1 \) never traverses. Let \( s \) (resp. \( t \)) the local index of this edge at \( u_b \) (resp. \( v_b \)). Let \( \mathcal{E}_{b_j} \) be the maximal prefix of execution where \( b_j \) is the only activated agent. Two cases are possible: either \( b_j \) eventually stops at some node in \( \mathcal{E}_{b_j} \) or \( \mathcal{E}_{b_j} \) can be infinitely extended.

Consider now a graph \( G = (V, E) \), where \( V = V_a \cup V_b \) and \( E = E_a \cup E_b \cup \{\{u_a, u_b\}, \{v_a, v_b\}\} \setminus \{\{u_a, v_a\}, \{u_b, v_b\}\} \) (refer to Figure 1). The local indices of \( \{u_a, u_b\} \) at \( u_a \) and \( u_b \) are \( i \) and \( s \), respectively. The local indices of \( \{v_a, v_b\} \) at \( v_a \) and \( v_b \) are \( j \) and \( t \), respectively. The other local indices remain unchanged.

Consider an execution of \( \mathcal{A} \) in \( G \) with only agents \( a_1 \) and \( b_1 \). Agents \( a_1 \) and \( b_1 \) can start at the same node as in \( \mathcal{E}_{a_i} \) and \( \mathcal{E}_{b_j} \), respectively. As a consequence, they can behave as in \( \mathcal{E}_{a_i} \) and \( \mathcal{E}_{b_j} \). Indeed, they do not visit the any common node so they act as if they were alone. In particular they never traverse edges \( \{u_a, u_b\} \) and \( \{v_a, v_b\} \) and the rendezvous never occurs, a contradiction.

**Lemma 2.** Any graph exploration requires \( \Omega(\log \Delta) \) bits in each whiteboard.

**Proof.** Consider an agent \( a \) that explores the graph. As \( a \) is oblivious, it can only use the whiteboard to store information about its traversal. To perform a deterministic traversal of the graph, in each node \( v \), \( a \) has to know whether all the edges incident to \( v \) have been traversed or not. So, \( a \) needs \( \Omega(\log(\delta_v)) \) bits in the whiteboard of \( v \) to perform its traversal and the lemma holds.

Note that Lemma 1 shows that in every rendezvous, at least one agent \( a \) explored the graph. However, this does not prove that the exploration (and by the way, the execution) eventually terminates. Now, termination is required to obtain a rendezvous. As shown in the next lemma, \( \Omega(\log L_{\text{max}}) \) additional bits in each whiteboard are necessary to implement termination.
Asymptotically Optimal Deterministic Rendezvous

The next lemma uses the notion of $\kappa$-regular graph. A graph $G$ is $\kappa$-regular if the degree of every node is equal to $\kappa$.

**Lemma 3.** In any deterministic rendezvous algorithm for general graphs, there are at least $\Omega(\log L_{\text{max}})$ bits in each whiteboard whose value depends on agent labels.

**Proof.** Let $\mathcal{A}$ be a deterministic rendezvous algorithm for general graphs. Consider a team of $k$ agents $a_0 \ldots a_{k-1}$ and a $\kappa$-regular connected graph $G$ of $n \geq k$ nodes.

Using $\mathcal{A}$, a deterministic rendezvous of $a_0 \ldots a_{k-1}$ can be done in $G$. By definition, once the rendezvous is achieved, $a_0 \ldots a_{k-1}$ are placed at a unique node $v_0$.

Nodes are anonymous. Moreover, agents are oblivious and can only communicate using whiteboards. So, the common decision to stop at $v_0$ is taken according to the whiteboard content of $v_0$. Moreover, the whiteboard of $v_0$ must be different of any other whiteboard. Hence, $\mathcal{A}$ allows, in particular, to elect a leader among the nodes of $G$, as stated in [3].

Every node $v$ of $G$ is anonymous, of degree $\kappa$, and every edge incident to $v$ is arbitrarily labeled with a value in $0 \ldots \kappa - 1$. So, the value in the whiteboard distinguishing the leader node $v_0$ can only be discriminated using the $\text{From()}$ functions and/or the labels of the agents$^\ast$. Clearly, the $\text{From()}$ functions are not sufficient because at the first activation, the $\text{From()}$ function of every agent returns $\perp$ and thereafter returns the number of the channel from which the agent arrived at a node. Therefore, the only way to ensure the unicity of some value in the whiteboard of $v_0$ is that the value that depends on some agent labels (remember that at least one of them is unique) and has $\Omega(L_{\text{max}})$ possible states.

$^\ast$Note that the leader can be selected using both $\text{From()}$ functions and agent labels like in our algorithm.

By Lemma 1, in any deterministic rendezvous algorithm, at least one agent performs a full traversal of the network. In our model, such a traversal is done in $\Omega(m)$ rounds where

![Fig. 1: Graph $G$ used in the proof of Lemma 1.](image-url)
\( m \) is the number of edges. Hence, the next theorem holds:

**Theorem 4.** Any deterministic rendezvous is done in \( \Omega(m) \) rounds where \( m \) is the number of edges.

By Lemmas 1, 2, and 3, follows:

**Theorem 5.** Any deterministic rendezvous algorithm requires \( \Omega(\log(\Delta) + \log(L_{\max})) \) bits of memory in the whiteboard of each node.

### 4. Algorithm

In this section, we propose a rendezvous algorithm working in the model given in Section 2. An informal description of our algorithm (Algorithm 1) is given in Subsection 4.1. In Subsection 4.2, we prove its correctness and study its complexity, showing then it is asymptotically optimal in both space and time complexities.

---

**Algorithm 1** Rendezvous of multiple agents in an undirected graph.

**Constants of an agent \( a \):**
- label\(_a\) : Agent label.

**Primitives at a node \( v \):**
- Go\((\text{edge})\) The agent moves through the specified edge.
- From() \( \in \{0,1,\ldots,\delta_v-1\} \cup \{\bot\} \)
  Returns the edge from which the agent comes, \( \bot \) otherwise.

**Variables at a node \( v \):**
- current \( \in \{0,1,\ldots,\delta_v-1\} \cup \{\bot\} \)
  Edge being currently explored by local host, initialized to \( \bot \).
- home : Boolean State if the node is currently considered as a home by some agent, initialized to \( \text{false} \).
- host : Label Agent of smallest label having visited this node, not initialized.

**Macros of an agent \( a \) at a node \( v \):**
- Explore\((\text{edge})\) = current \( \leftarrow \) edge; Go\((\text{edge})\)
- Next() = (From() + 1) \mod \delta_v

**Predicates at a node \( v \):**
- FirstStep = From() = \( \bot \)
- NewNode = current = \( \bot \) \lor label_\( a \) < host
- OwnNode = current \neq \( \bot \) \land host = label_\( a \) \land From() = \( \bot \)
- IAmAGuest = current \neq \( \bot \) \land host < label_\( a \)
- Cycle = current \neq From()
- HostHome = home = \( \text{true} \)
- End = HostHome \land (Next() = 0)

**Guarded commands of an agent \( a \) at a node \( v \):**
- StartAsHost : NewNode \land FirstStep \implies host \leftarrow label_\( a \); home \leftarrow true; Explore(0)
- ExploreNextNode : NewNode \land \neg FirstStep \implies host \leftarrow label_\( a \); home \leftarrow true; Explore(Next())
- FollowAsGuest : IAmAGuest \land \neg HostHome \implies Go(From())

---

#### 4.1. Overview

In the following, we call *host* the unique agent with the smallest label. All other agents are called *guests*. The principle of the algorithm is to group all agents at the host home.

Our algorithm works in two phases: the *traversal* and *assembling* phases. Each agent starts by a *traversal* phase during which it builds a spanning tree rooted at its home and
writes its label in all the whiteboards. This phase is only aborted once the agent learns that it is not the host. In this case, it switches to the assembling phase. Note that the host always completes its traversal and this traversal terminates at its home. The assembling phase is performed by the guests only. In this phase, a guest follows the edges it believes to be in the host spanning tree in order to reach the host home.

More precisely, the host builds a spanning tree rooted at its home by writing the home, current, and host variables in all the whiteboards. At the beginning, no agent knows who is the host, what is the host label, and where is the host home. So, every agent acts as it is the host until it has evidence of the contrary. To obtain such an evidence, every agent writes its label into the whiteboards during the spanning tree construction until discovering a node marked with a smaller label. In this case, it switches to a second phase: it uses the information in the whiteboards to follow a path of the host spanning tree to its home. However, guests may make mistakes either (1) by wrongly considering another guest agent (with a smaller label) as the host agent or (2) by following a cycle in a tree still under construction. Now, each time an agent is activated at a node where the whiteboard contains a label that is greater than its own, it considers the node as unvisited and overwrites the whiteboard. Also, the host’s writings are never overwritten because its label is unique and minimum. As a consequence, the traversal of the host eventually terminates in its home: the host stays forever in its home, a spanning tree rooted at the host home is available, and the host label is written in all host variables. From that point on, no mistake is possible anymore. That is, all guests agree on the host label and follow the host spanning tree stored into the whiteboards until reaching the host home: the rendezvous is eventually achieved.

We now describe the behavior of the host and the guests separately.

The host. The host $h$ detects its first activation thanks to the From() function that returns $\perp$. In this case, it marks its home by writing information in the whiteboard: a home variable and a host variable are set to true and label$_h$ respectively. Then, starting from its home, $h$ performs a depth-first traversal to construct the spanning tree and broadcast its label. To that end, it initializes the current variable of the whiteboard to 0 and leaves the node by edge 0. Upon arriving into an unvisited node, either the whiteboard is empty (host = $\perp$) or it has been written by a guest $g$ (host = label$_g$ with label$_g$ < label$_h$). In both cases, $h$ writes (or overwrites) the whiteboard by assigning the host variable to its label label$_h$ and the home variable to false (this latter assignment erases the guest homes). Then, $h$ continues its traversal by assigning current to From() + 1 (mod $\delta_v$) and leaving the node by the current edge pointed by current. Upon arriving into an already visited node (i.e., a node where host = label$_h$), two cases are possible for $h$: (i) either current $\neq$ From(), (ii) or current = From(). In case (i), $h$ has followed a cycle, so it returns to the node it comes from (using From()). In case (ii), $h$ has been backtracked because the traversal from the edge it comes is done. Hence it increments the current variable and then leaves the node by the current edge in order to continue the traversal. Using this method, $h$ eventually terminates its traversal at its home: when it arrives in a node $v$ where host = label$_h$, home = true, and current = $\delta_v$ − 1. In this case, a spanning tree rooted at its home is described by the current variables in all the whiteboards and every host variable is set to label$_h$. 
The guests. Consider any guest \( g \). Agent \( g \) acts as the host (i.e. it performs a traversal of the graph) until it is activated at a node where \( \text{host} < \text{label}_g \). Then, \( g \) switches to the assembling phase: each time \( g \) is activated, it decides that it is a guest because \( \text{host} < \text{label}_g \) and, as a consequence, it leaves the node by the edge pointed by \( \text{current if home} \neq \text{true} \), otherwise it stays at the node (it believes to be in the host home). Using this mechanism,\( g \) eventually definitively stops at the host home because the \( \text{current} \) variables eventually describe a spanning tree rooted at the host home which is eventually the only node where \( \text{home} = \text{true} \) holds.

Note that a guest may arrive at nodes written by agents having the same label. In this case, the guest believes to be in a node that it already visited (because it is oblivious). This situation does not make the algorithm fails because the nodes are eventually visited by the host and, consequently, marked with the host label which is unique.

4.2. Correctness

We first consider the behavior of the host agent \( h \). From the code of Algorithm 1, we can remark that the host label being minimal, \( h \) only executes the traversal phase.

Below, we show that the traversal of the host \( h \) can only terminate at its home.

Lemma 6. When the traversal of \( h \) terminates, \( h \) is located at its home \( v_h \).

Proof. Assume, by the way of contradiction, that the traversal of \( h \) terminates at a node \( v \neq v_h \).

When \( h \) is activated in \( v \) for the first time, \( \text{From}() \) returns a value different of \( \bot \) and, as a consequence, \( h \) sets the variables \( \text{home}_v \) and \( \text{host}_v \) to \( \text{false} \) and \( \text{label}_h \), respectively (action ExploreNewNode). Moreover, if \( \text{current}_v = \bot \), \( h \) sets \( \text{current} \) to a value different of \( \bot \) (actually, the value returned by \( (\text{From}() + 1) \mod \delta_v \)).

From that point, no agent other than \( h \) can write into the whiteboard of \( v \) and \( h \) never overwrites \( \text{home}_v \) and \( \text{host}_v \). Moreover, \( h \) never sets \( \text{current}_v \) to \( \bot \).

Hence, when \( h \) terminates its traversal at \( v \), \( \neg \text{End} \) and \( \text{OwnNode} \) hold because \( \text{home}_v = \text{false} \), \( \text{current}_v \neq \bot \), \( \text{host}_v = \text{label}_h \), and \( \text{From}() \neq \bot \).

As a consequence, either action ExploreNextEdge or action UndoCycle of \( h \) is enabled: \( h \) cannot terminate its traversal at \( v \), a contradiction. Hence, \( h \) can only terminate at its home and the lemma holds.

Lemma 7. If \( h \) terminates its traversal at its home \( v_h \), then the following conditions hold:

1. \( \text{home}_{v_h} = \text{true} \).
2. \( \text{host}_{v_h} = \text{label}_h \).
3. \( \text{current}_{v_h} = \text{From}() = \delta_v - 1 \).

Proof. First, \( h \) starts its traversal by action StartAsHost. After executing this action, \( \text{From}() \) always returns a value different of \( \bot \).

Assume now that \( h \) terminates its traversal in its home \( v_h \). Then, every action of \( h \) at \( v_h \) are disabled.
As \( \text{From}() \neq \bot \), predicate \(~F\text{irstStep}\) holds. So, predicate \( \text{NewNode} \) does not hold otherwise action \( \text{ExploreNewNode} \) is enabled. As a consequence, \( \text{host}_v = \text{label}_v \) (condition (2)) and \( \text{current}_{\text{v}} \neq \bot \).

As \( \text{From}() \neq \bot \), \( \text{host}_{\text{v}} = \text{label}_{\text{v}} \), and \( \text{current}_{\text{v}} \neq \bot \), predicate \( \text{OwnNode} \) holds. So, predicate \( \text{Cycle} \) does not hold otherwise action \( \text{UndoCycle} \) is enabled. As a consequence, \( \text{current}_{\text{v}} = \text{From}() \neq \bot \).

Finally, as predicate \( \text{OwnNode} \land \sim \text{Cycle} \) holds, predicate \( \text{End} \) holds otherwise action \( \text{ExploreNextEdge} \) is enabled. As a consequence, \( \text{home}_{\text{v}} = \text{true} \) (condition (1)) and \( \text{From}() = \delta_{\text{v}} - 1 \). Hence, \( \text{current}_{\text{v}} = \text{From}() = \delta_{\text{v}} - 1 \) (condition (3)) and the lemma holds.

\[ \square \]

In the following, Lemmas 8 to 10 show that \( h \) traverses each edge at most 4 times. As a corollary of this result, we can state that \( h \) eventually terminates its traversal.

We say that a node \( v \) designates the link from \( v \) to \( v' \) for agent \( a \) if and only if \( \text{host}_v = \text{label}_v \) and \( \text{current}_v = i \) where \( i \) is the local index of \( \{v, v'\} \) at node \( v \). Moreover, an agent \( a \) explores the edge \( \{v, v'\} \) from \( v \) if it is activated at node \( v \) and, consequently, it executes \( \text{current}_v \leftarrow i \) where \( i \) is the local index of \( \{v, v'\} \) at node \( v \).

**Lemma 8.** For every node \( v \), for every edge \( e \) incident to \( v \), the host agent \( h \) explores \( e \) from \( v \) at most once.

**Proof.** We prove this lemma by induction on the order \( h \) visits the nodes.

- **Base Case:** The first node visited by \( h \) is its home \( v_h \). When \( h \) is activated for the first time in \( v_h \), \( \text{From}() \) returns \( \bot \) and predicate \( \text{NewNode} \) holds because the label of \( h \) is unique and minimal. So, \( h \) executes action \( \text{StartAsHost}: \text{host}_{v_h}, \text{home}_{v_h}, \text{current}_{v_h} \) are set to \( \text{label}_{v_h}, \text{true}, \) and \( 0 \), respectively. From that point, no agent other than \( h \) can write into the whiteboard of \( v_h \) and \( h \) never overwrites \( \text{home}_{v_h} \) and \( \text{host}_{v_h} \). Moreover, \( h \) can only increment \( \text{current}_{v_h} \) one by one (modulo \( \delta_{v_h} \)) using action \( \text{ExploreNextEdge} \) until \( \delta_{v_h} - 1 \). After that, \( h \) never more modifies \( \text{current}_{v_h} \) and the induction holds in this case.

- **Induction Hypothesis:** Let \( k > 0 \). Let \( v_k \) be the \( k^{th} \) node visited by \( h \). Assume that for every node \( v_i \) visited by \( h \) before \( v_k \), for every edge \( e_i \) incident to \( v_i \), \( h \) explores \( e_i \) from \( v_i \) at most once.

- **Inductive Step:** Assume, by the way of contradiction, that \( h \) explores an edge from \( v_k \) at least twice.

When \( h \) is activated for the first time in \( v_k \), \( \text{From}() \) returns a local index \( i \) (with \( i \neq \bot \)) and predicate \( \text{NewNode} \) holds because the label of \( h \) is unique and minimal. So, \( h \) executes action \( \text{ExploreNewNode}: \text{host}_{v_k}, \text{home}_{v_k}, \text{current}_{v_k} \) are set to \( \text{label}_{v_k}, \text{false}, \) and \( (i + 1) \mod \delta_{v_k} \), respectively. From that point, no agent other than \( h \) can write into the whiteboard of \( v_k \) and \( h \) never overwrites \( \text{home}_{v_k} \) and \( \text{host}_{v_k} \). Moreover, \( h \) can only increment \( \text{current}_{v_k} \) one by one (modulo \( \delta_{v_k} \)) using action \( \text{ExploreNextEdge} \). Consequently, before exploring an edge for the second time from \( v_k \), \( h \) must explore the edge \( \{v_k, v_\ell\} \) (with \( \ell < k \)) from \( v_k \) where \( \{v_k, v_\ell\} \)
is the edge of local index $i$ at node $v_k$.

So, let us focus on edge $[v_k, v]$. Let $j$ be the local index of $[v_k, v]$ at node $v$. When $v_k$ sends $h$ to $v$, for the first time, From() returns a value different of $j$ because $v_k$ was never visited by $h$. From the code of Algorithm 1, either ExploreNewNode or ExploreNextEdge was executed to send $h$ to $v_k$. After executing one of these actions, host$_v$ and current$_v$ are equal to label$_h$ and $j$, respectively. From that point, no agent other than $h$ can write into the whiteboard of $v$, $h$ never overwrites host$_v$, and current$_v$ remains equal to $j$ until receiving $h$ from $v_k$.

When $h$ is activated for the first time in $v_k$, either $(i + 1) \mod \delta_{v_k} = i$ (that is the degree of $v_k$ is one) or $v_k$ sends $h$ to another node than $v$. In this latter case, until $v_k$ sends back $h$ to $v$, From() $\neq i$ when $h$ is activated in $v_k$, indeed each time $v_k$ receives $h$ from a node different of $v_k$ it directly backtracks $h$ to the node.

Thus, in all cases, current$_{v_k}$ must be set to $i$ so that $v_k$ sends $h$ to $v$. After this sending, current$_{v_k}$ remains to $i$ until $v_k$ receives $h$ again from $v$. Moreover, when $v$ receives $h$ from $v_k$, From() = $j$ = current$_{v_k}$ and two cases are possible: either (1) $j = \delta_{v_k} - 1$ and $v_k$ is the home of $h$ or (2) current$_{v_k}$ is incremented and $h$ is sent to current$_v$. In the former case, the traversal of $h$ terminates and, as a consequence, no edge incident to $v_k$ is explored from $v_k$ at least twice, a contradiction. In the latter case, current$_{v_k}$ $\neq j$ forever by induction hypothesis and while $v$ does not receive $h$ from $v_k$, From() $\neq j$ and consequently, $v$ does not send $h$ to $v_k$. To sum up, $v_k$ cannot send $h$ to $v$ before $v$ sends $h$ to $v_k$ and conversely. As a consequence, current$_{v_k}$ remains fixed forever, which is a contradiction: the induction holds in this case.

\[ \square \]

**Lemma 9.** Let $e$ be an edge between nodes $v$ and $v'$. If $h$ leaves $v$ or $v'$ by edge $e$, then $v$ or $v'$ designates $e$ for $h$.

**Proof.** Assume, by the way of contradiction, that $h$ leaves $v$ or $v'$ by $e$ while neither $v$ nor $v'$ designates $e$ for $h$. Consider the first time, $h$ does that and, without loss of generality, assume that $h$ leaves $v$. Then, from the code of Algorithm 1, when $h$ is activated to leave $v$, From() = $i$ where $i$ is the local index of $e$ at $v$. So, this means that, $h$ previously arrived at $v$ from edge $e$ and, by hypothesis, when $h$ leaves $v'$ to $v$, (1) $host_v = label_h$ and current$_e = i$ or (2) $host_{v'} = label_h$ and current$_{v'} = j$ where $j$ is the local index of $e$ at node $v'$.

In the former case, we obtain a contradiction because when $h$ arrives in node $v$, no other agent modified $h'$ writings and either (i) $i = \delta_v - 1$, $v$ is the home of $h$ (as a consequence host$_{v'} = label_h$), and the traversal of $h$ is terminated or (ii) current$_e$ is incremented and $h$ leaves $v$ through the edge of index current$_e$. If current$_e = i$, $v$ designates $e$ for $h$ when $h$ leaves $v$ by $e$, a contradiction. Otherwise, $h$ leaves $v$ through another edge than $e$.

In the latter case, no other agent has modified $h'$ writings when $h$ arrived in $v$ from $v'$. So, $v'$ still designates $e$ for $h$ when $h$ is activated to leave $v$ by $e$, a contradiction. \[ \square \]

**Lemma 10.** Host agent $h$ traverses an edge $e$ from node $v$ at most twice.

**Proof.** Let $i$ be the local index of $e$ at node $v$. When $h$ decides to traverse an edge $e$ from
node \(v\), the two following cases are possible:

- \(\text{current}_v = i\). In this case, \(\text{current}_v\) remains equal to \(i\) until \(v\) receives \(h\) by edge \(e\) (no other agent can overwrite \(h\) writings). When \(v\) receives \(h\) by edge \(e\) either (i) \(i = \delta_v - 1\), \(v\) is the home of \(h\), and the traversal of \(h\) is terminated or (ii) \(\text{current}_v\) is incremented to a value different of \(i\) by Lemma 8, \(\text{current}_v \neq i\) forever. Hence, this case occurs only once.

- \(\text{current}_v \neq i\). In this case, \(\text{current}_v = j\) where \(v'\) is the other node incident to \(e\) and \(j\) is the local index of \(e\) at node \(v'\) by Lemma 9. When \(v'\) receives \(h\) from edge \(e\), then either (i) \(j = \delta_v - 1\), \(v'\) is the home of \(h\), and the traversal of \(h\) is terminated or (ii) \(\text{current}_v\) is incremented to a value different of \(j\) and by Lemma 8, \(\text{current}_v \neq j\) holds forever. Thus, this case can occur only once.

Hence, we can conclude that \(h\) traverses \(e\) from node \(v\) at most twice.

**Corollary 11.** Host agent \(h\) terminates its traversal at its home in at most \(8m\) rounds where \(m\) is the number of edges.

**Proof.** By Lemma 10, \(h\) can leave any node \(v\) at most \(2\delta_v\). So, during its traversal, \(h\) executes at most \(\sum_{v \in V} 2\delta_v\) edge-crossings. By the Handshaking Lemma [15], \(\sum_{v \in V} 2\delta_v = 4m\). So, \(h\) executes at most \(4m\) edge-crossings. Now, each edge-crossing is performed in at most one round and each edge-crossing is preceded by an activation in a node performed in at most one round. Hence, \(h\) performed its traversal in at most \(8m\) rounds and, by Lemma 6, this traversal terminates at its home which concludes the proof.

Let us consider now the subgraph of \(G\) made of the “traces” of \(h\) during its traversal. In the next lemmas (Lemmas 12 and 13), we show that this subgraph is a spanning tree of \(G\). More formally, let \(v_h\) be the home of \(h\) and \(V' \subseteq V\) be the subset of nodes such that \(\forall v \in V'\), \(\text{host}_v = h\) when the traversal of \(h\) is terminated. Let \(E'\) be a set of directed edges such that \((v', v) \in E'\) if and only if \(v' \in V' \setminus \{v_h\}\) and \(\text{current}_{v'}\) designates \(v', v\). Let \(T = (V', E')\). An *in-tree* [4] is an directed tree in which a single vertex (called *root*) is reachable from every other one.

**Lemma 12.** \(T\) is an in-tree.

**Proof.** Assume, by the way of contradiction, that \(T\) is not an in-tree. Then, two cases are possible:

- \(T\) contains a directed cycle \(v_0, \ldots, v_c\). The traversal of \(h\) being sequential, we can consider the last node, say \(v_c\), where the \(\text{current}\) variable was modified to create the cycle. First, by definition of \(T\), \(v_c\) is not the home of \(h\). Then, after \(v_c\) designates the edge \(e\) of the cycle for \(h\), \(h\) leaves the node through \(e\). Upon arriving in the destination node, \(v_{c+1}\), either (1) \(v_{c+1}\) designates \(e\) for \(h\) (in this case the length of the cycle is two) or (2) \(v_{c+1}\) designates for \(h\) an edge that is not \(e\).
In the former case, if the degree of $v_{i+1}$ is one, then $h$ is sent to by $v_i$, and upon arriving in $v_i$, the degree of $v_i$ is more than one (otherwise $v_i$ or $v_{i+1}$ is the home of $h$ because there are exactly two nodes in the graph) and $current_h$ is incremented, a contradiction. If the degree of $v_{i+1}$ is more than one, then $current_{v_{i+1}}$ is incremented, a contradiction.

In the latter case, upon arriving in $v_{i+1}$, $h$ is sent to by $v_i$. Upon arriving in $v_i$, if the degree of $v_i$ is one, then $h$ is sent back to $v_{i+1}$ and the traversal of $h$ never terminates, a contradiction to Corollary 11. Otherwise, $current_h$ is incremented, a contradiction.

- $T$ has several connected components. Then, as $T$ cannot contain a cycle, we can deduce that some node $v$ in $T$ designates an edge $e$ for $h$ linking $v$ to a node $v'$ that is not in $T$. After $v$ designates $e$ for $h$, $h$ was sent to $v'$ through $e$. Once in $v'$, $h$ wrote its label in the whiteboard if it was not already written. Now, from that point, no agent other than $h$ can write into the whiteboard of $v'$ and $h$ never overwrites $host_{v'}$. So, $v'$ belongs to $T$, a contradiction.

Hence, $T$ contains no directed cycle and only one connected component: $T$ is an in-tree. □

**Lemma 13.** $T$ is a spanning in-tree of $G$.

**Proof.** Assume, by the way of contradiction, that $T$ is not a spanning in-tree of $G$. By lemma 12, $T$ is a in-tree. So, this means that there exists a node $v$ in $G$ that is not in $T$. That is, once the traversal of $h$ is terminated, the $host$ variable of the node does not contain the label of $h$ and, as a consequence, has been never visited by $h$. As $h$ visits at least one node (its home) and $G$ is connected, there is at least one node $v$ unvisited by $h$ that is neighbor of a node $v'$ visited by $h$.

The first time $v'$ is visited by $h$, $v'$ received $h$ from an edge $e = \{v', v''\}$. Let $i$ be the local index of $e$ at $v'$. As $v'$ was not already visited by $h$, $v'$ does not designate any edge for $h$ and, by Lemma 9, $v''$ designates $e$ for $h$. From that point, $v''$ designates $e$ for $h$ until receiving $h$ through $e$, i.e., until $h$ is activated in $v''$ and $current_{v''} = \text{From}(i)$.

Moreover, the first time $v'$ is visited by $h$, $host_{v'}$ and $current_{v'}$ are set to $label_h$ and $i + 1 \mod \delta_{v'}$, respectively. From that point, $host_{v'}$ is fixed and only $h$ can modify $current_{v'}$. Now, $v$ is never visited from $v'$ and $current_{v'}$ can only be incremented modulo $\delta_{v'}$. So, $h$ is never sent back to $v''$ through $\{v', v''\}$. Thus, the value of $current_{v'}$ is fixed and each time $h$ is activated in $v'$, $current_{v'} \neq \text{From}(i)$.

Inductively, we can deduce that each time $h$ is activated in its home $v_h$, $current_{v_h} \neq \text{From}(i)$, which contradicts Lemmas 6, 7, and Corollary 11. □

We are now ready to state our main result (Theorem 15). It requires the following technical result:

**Lemma 14.** A rendezvous between all agents occurs within at most $2(n - 1)$ rounds after the host agent $h$ terminates its traversal.
Proof. First, after $h$ terminates its traversal, the variable $home$ is equal to true at its home by Lemma 7.

Then, when $h$ is activated at a node, $\text{From}()$ returns $\bot$ only if $h$ is at its home. So, each time $h$ visits for the first time a node $v$ that is not its home it executes action $\text{ExploreNewNode}$: $home_v$ is set to $false$ and $host_v$ is set to $label_h$. After that, no agent other than $h$ can write into the whiteboard of $v$ and $h$ never overwrites $home_v$ and $host_v$. So, $home_v = false$ and $host_v = label_h$ forever.

Now, by Lemma 13, all nodes are visited by $h$ before the end of its traversal. So, after $h$ terminates its traversal, for every node $v$, $home_v = true$ if and only if $v$ is the home of $h$.

Moreover, by Lemma 13, for every node $v$, $host_v = label_h$ and $current_v$ designates an edge of a spanning tree rooted at the home of $h$.

So, for each guest agent $g$, while $g$ is not at the home of $h$, $g$ is enabled and when activated, it moves toward the spanning tree $T$ to the home of $h$. The height of $T$ is bounded by $n-1$. So, after crossing at most $n-1$ edges and $n-1$ nodes, any guest is at the home of $h$.

Hence, at most $2(n-1)$ rounds after the host agent $h$ terminates its traversal, the rendezvous is achieved.

Theorem 15. Algorithm 1 is a deterministic rendezvous algorithm satisfying the following properties:

(1) It allows agents to meet in at most $8m + 2(n-1)$ rounds.
(2) It is asymptotically optimal in rounds.
(3) It is asymptotically optimal in space.

Proof.

• Property (1): Immediate from Corollary 11 and Lemma 14.
• Property (2): As $G$ is connected, we have $n \leq m$. Hence, from Property (1), we know that the rendezvous occurs in $O(m)$ rounds, which is asymptotically optimal by Theorem 4.
• Property (3): By checking Algorithm 1, we can remark that the space requirement of Algorithm 1 matches the result of Theorem 5. Hence, Algorithm 1 is asymptotically optimal in space.

5. Some necessary conditions

In this section, we prove that some of the conditions we used to build the model described in Section 2 are necessary to deterministically solve the rendezvous problem.

In Section 2, we formalized the deterministic rendezvous problem in a model where few conditions have been added to the initial weak scenario presented in the introduction (Section 1). This set of conditions is:

(1) Agents are labeled and nodes are provided with whiteboards.
(2) There is a strict extremum label assigned to a unique agent.
(3) Edges have local indices at nodes.
Below, we show that the above three conditions are necessary.

Assume by contradiction that one of these conditions can be ignored.

(1) Removing agent labels or node whiteboards brings back to the anonymous rendezvous problem which has been proved not to be solvable in this setting [3].

(2) Removing the strict extremum on agent labels or the unicity of this label forbids to particularize an agent. As shown in the proof of Lemma 3, the achievement of a deterministic rendezvous is subjected to the existence of a unique agent that decides of the meeting point. Hence, this property is also necessary.

(3) Removing edge local indices transforms agent moves into a random walk. Then the algorithm is not truly deterministic.

So, removing any of these conditions makes the rendezvous problem impossible to be solved by a deterministic algorithm. Thus, this set of conditions is minimal.

6. Conclusion

We considered the deterministic rendezvous problem of mobile agents in simple undirected anonymous connected graphs. We provided a model with several minimal hypothesis. Then, we proved asymptotical bounds, both in memory size and number of rounds, for any deterministic rendezvous algorithm in our model. We gave an algorithm that is asymptotically optimal in both space and round complexities.

A natural extension of this work would be to find exact bounds and a solution that exactly matches these bounds. Another future work would be to investigate rendezvous in directed connected graphs. Such graphs can model wireless networks where antenna ranges are heterogeneous. In directed connected graphs, a node may be not reachable from an other and conversely. Hence, rendezvous is not always achievable. Therefore, an interesting question arises: “What is the maximal class of directed graphs that admit a solution?”.

We conjecture that the class of graph containing one sink component [18] (n.b., a sink component is a subgraph that any agent cannot leave) is a good candidate. Of course, optimal bounds for the rendezvous in such graphs is also an open question.

References


Asymptotically Optimal Deterministic Rendezvous


