Distributed Exploration of Anonymous Graphs by Multiple Agents[*]

Shantanu Das[†] Paola Flocchini[†] Shay Kutten[‡] Amiya Nayak[†] Nicola Santoro[§]

Abstract

We consider the problem of exploration and mapping of an unknown environment modelled as a graph, by multiple identical mobile agents that are dispersed among the nodes of the graph. The objective is for each agent to build an identically labelled map of the graph; we call this the *Labelled Map Construction* problem. The problem is of much practical importance because having such a map facilitates the navigation and coordination among the agents when they are lost in an unknown environment. We give deterministic solutions to the problem under the weakest possible setting, assuming the agents only have the knowledge of either the size of the graph or the number of agents present.

Keywords: Labelled Map Construction, Graph Exploration, Leader Election, Rendezvous, Anonymous, Asynchronous, Mobile Agents.

1 Introduction

The problem of exploring and mapping an unknown environment has been extensively studied due to its various applications in different areas. Some examples are navigating a robot through a terrain containing obstacles, finding a path through a maze, or searching a computer network using mobile software agents. The environment to be explored is often modelled as a graph, (or a digraph) where a single entity (called an agent or robot) starting at one of the nodes of the graph, traverses through all the edges of the graph and returns back to the starting point, constructing a map of the graph, in the process. If the nodes of the graph are marked with unique labels then the agent can uniquely identify each node it visits, so that a simple depth-first traversal of the graph is enough to construct a map of the graph. In the absence of such labels however, the task is more difficult, but can still be achieved if the agent is supplied with a marking device for marking the nodes during exploration.

Now, suppose instead of a single agent, there are several identical agents starting at different nodes of the graph and each of them is trying to explore the graph. We want each agent to build a map of the graph and further the maps obtained by the agents should be consistent with one another, for example if a node is labelled as node-1 in one agent's map, it should be labelled the same in the other agent's maps. The objective is to convert an unknown and anonymous environment with dispersed agents into a mapped and totally labelled environment where the agents are in agreement with each other.

The agents are identical in all respect—they have the same capabilities and follow the same protocol. However, we do not assume the presence of any synchronization between the agents (i.e. the

[*]A preliminary version of this paper appeared in the proceedings of 12th Colloquium on Structural Information and Communication Complexity (SIROCCO), France 2005.

[†]School of Information Technology and Engineering, University of Ottawa, Canada. Email:{shantdas,flocchin,anayak}@site.uottawa.ca

[‡]Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology, Israel. Email: kutten@ie.technion.ac.il

[§]School of Computer Science, Carleton University, Canada. Email: santoro@scs.carleton.ca

agents do not share the same clock). The only means of communication among the agents is using a limited amount of shared memory available at the nodes of the graph. So, each node of the graph contains a whiteboard where any visiting agent can read or write information; access to the whiteboard is restricted by fair mutual exclusion. The goal for the agents is to construct a labelled map of the unknown graph, the same for all agents, so to be in complete agreement about their environment. We call this problem *labelled map construction* (LMC).

The motivation for modelling the environment as a unlabelled graph, comes from many practical considerations. For example, consider a robot traversing a *graph-like* world, where the edges are roads and the nodes are the intersections; the robot may not have enough sensory capabilities to uniquely identify a node that it visits. As another example, consider a software agent traversing a network, where the identification field of the nodes (i.e. the network hosts) are kept hidden from the visiting agents, say due to security considerations. Thus, from the viewpoint of the agents, we can assume the nodes of the graph to be unlabelled (*anonymous*), so that all nodes of same degree look the same to an agent. Clearly, in order to explore such an anonymous graph, the agents need to somehow mark the nodes (by writing on the whiteboards), so that previously visited nodes can be identified on subsequent visits. However, having multiple identical agents creates problems here: marks made by one agent could be indistinguishable from those made by another; different agents might mark in the same way different nodes. Thus, it is not clear whether the agents can successfully map an anonymous graph.

This problem, as we show, is closely related to some others basic problems, like *Agent Election*, *Labelling* and *Rendezvous* in such a way that solving any one of these problems will allow us to solve all the others too. In this paper, we design efficient and generic protocols that can solve these problems, irrespective of the graph topology, where the cost of the algorithm is measured in terms of the total number of moves (or, edge traversals) made by the agents.

1.1 Our Results

We first show that solving the *Labelled Map Construction* problem is as difficult as solving other related problems like *Agent Election*, *Labelling* and *Rendezvous*. This allows us to determine the necessary conditions that need to be satisfied for solving the LMC problem in an arbitrary graph. For instance, the agents need to have the prior knowledge of the value of n (the number of nodes in the graph) or else k (the number of agents). However even with this knowledge it is not always possible to solve the problem in cases where $\gcd(n, k) > 1$. But if the value of n and k are co-prime to each-other, we can always have a guaranteed solution to the *Labelled Map Construction* problem (assuming the knowledge of either n or k).

We first present a protocol that will allow a team of k anonymous agents scattered in an unknown unlabelled graph of n nodes and m links to construct a map of the graph, and elect a leader among the agents, using no more than $O(m.k)$ edge traversals. We then show that the complexity of this algorithm can be improved to $O(m \log k)$, when both n and k are known a-priori to the agents (or at least the value of $\gcd(n, k)$ is known along with either n or k).

Our results are an improvement on the previous results for exploration, leader election or *Rendezvous*, which solve these problems either under additional constraints such as the presence of unique identities for the agents [6, 9, 24] or the presence of *sense of direction* [4], or for specific topologies (e.g. rings [13], trees [14]). Thus, our protocols are more generic and are applicable in a wider range of settings. Further, our algorithms are deterministic and include mechanisms for explicit termination detection.

In section 2.2 we formally describe the LMC and other related problems and discuss about their relationship. In section 3, we give an algorithm for collaborative exploration of the graph by multiple agents, that uses only a single bit of whiteboard memory and makes $O(m)$ moves. We then extend this algorithm, in section 4, to construct a spanning tree of the graph, elect a leader among the agents and build a uniquely labelled map of the graph. We show the correctness of our protocol in section 5 and finally in section 6 we show how the efficiency of the algorithm can be improved in the presence of additional knowledge available to the agents.

1.2 Related Work

Most of the previous work on exploration of unknown graphs has been limited to single agent exploration. In a labelled graph, each node is uniquely identifiable and thus, it is always possible to explore and map the graph, by just traversing the graph. Studies on exploration of *labelled* graphs (or digraphs), have emphasized minimizing the cost of exploration in terms of either the number of moves (edge traversals) or, the amount of memory used by the agent (e.g., see [1, 10, 8, 21, 22]).

Exploration of *anonymous* graphs is possible only if the agents are allowed to mark the nodes in some way; except when the graph has no cycles (i.e. the graph is a tree [14, 11]). For exploring arbitrary anonymous graphs, various methods of marking nodes have been used by different authors. Bender *et al.* [5] proposed the method of dropping a pebble on a node to mark it and showed that any strongly connected directed graph can be explored using just one pebble, if the size of the graph is known and using $O(\log\log n)$ pebbles, otherwise. Dudek *et al.* [12] used a set of distinct markers to explore unlabeled undirected graphs. Yet another approach, used by Bender and Slonim [6] was to employ two cooperating agents, one of which would stand on a node, thus marking it, while the other explores new edges. The whiteboard model, which we use here, has been earlier used by Fraigniaud and Ilcinkas [15] for exploring directed graphs and by Fraigniaud et al. [14] for exploring trees. In [11, 15] the authors focus on minimizing the amount of memory used by the agents for exploration (they however do not require the agents to construct a map of the graph).

There have been very few results on exploration by more than one agent. As mentioned earlier, a two agent exploration algorithm for directed graphs was given in [6], whereas Fraigniaud et al. [14] showed how k agents can explore a tree. In both these cases however, the agents are co-located (i.e. they start from same node at the same time) and they have distinct identities.

The other problems which have been studied in the similar setting of mobile agents dispersed in a graph are the problems of Rendezvous (i.e. gathering the agents in a single node) and Agent Election (electing a leader among the agents). The research on the *Rendezvous* problem is rather extensive; for a recent account see [2]. However, most of these results are obtained using probabilistic algorithms. Among deterministic solutions to the problem, the investigations of Yu and Yung on synchronous graphs of known topology [24] and of Dessmark *et al.* on synchronous rings and graphs [9] are limited to agents with *distinct* labels. In the context of anonymous agents in an unlabelled network, the only known results are those of Flocchini *et al.* and of Kranakis *et al.* on ring networks using pebbles [13, 20], and those of Barriere *et al.* on graphs with *sense of direction* [4]. In [3], Barriere *et al.* consider solutions to the agent election problem in presence of distinct but incomparable agent labels.

The results obtained in [4] are closely related to our results. In that paper, the authors solve the rendezvous and agent election problem in a setting similar to our model, but with the additional assumption that the edge-labelling on the graph provides a *sense of direction* to the agents[1].

Our work is also related to some of the classical results in the traditional distributed computing model (where the computing entities are stationary processors communicating through message passing). In that model, Gallager, Humblet and Spira [18] gave a distributed algorithm for leader election and spanning tree construction, in *labelled* graphs. Korach, Kutten and Moran [19] showed

[1]Notice that this implies that the agents can easily determine the size of the graph and thus this is a stronger model than the one considered in this paper.

that the leader election problem is closely related to graph exploration and they proposed the territory acquisition approach for electing a leader in arbitrary (but labelled) graphs. Sakamoto [23] considered anonymous networks and has given an algorithm that builds a spanning forest of the graph under a variety of initial conditions.

2 Model and Problems

2.1 The Model

The environment to be explored by the agents is a simple undirected connected graph $G = (V, E)$ having $n = |V|$ nodes. The labels (if any) on the nodes are invisible to the agents, so that the nodes are anonymous to the agents. However, an agent visiting a node can distinguish among the various edges incident at that node[2]. In other words, the edges incident to a node in the graph are locally labelled with distinct port numbers. However, this labelling is totally arbitrary and there is no coherence between the labels assigned to edges at the various nodes. Without loss of generality, we assume that the links incident at a node u are labelled as $1, 2, 3, \ldots, d(u)$, where $d(u)$ is the degree of that node. Note that each edge $e = (u, v)$ has two labels, one for the link or port at node u and another for the link at node v. We denote the former label as $l_u(e)$ and the later as $l_v(e)$; these two labels are possibly different. The edge labelling of the graph G is specified by $\lambda = \{l_v : v \in V\}$, where for each vertex u, $l_u : \{(u, v) \in E : v \in V\} \to \{1, 2, 3, \ldots, d(u)\}$ defines the labelling on its incident edges. We denote by Δ, the maximum degree of a node in the graph.

There are k agents and each agent starts from a distinct node of the graph, called its *homebase*. The agents have computing and storage capabilities, execute the same protocol, and can move from a node to the neighboring node in G. After moving from u to v, an agent has the label $l_u(u, v)$ of the edge from which it departed, as well as the label $l_v(v, u)$ of the edge from which it arrived. Whenever an agent reaches a node, it can communicate directly with the other agents present in that node.

The agents are *anonymous* in the sense that they do not have distinct names or labels. They execute a protocol (the same for all agents) that specifies the computational and navigational steps. They are *asynchronous*, in the sense that every action they perform (computing, moving, etc.) takes a finite but otherwise unpredictable amount of time.

An agent can communicate with other agents by leaving a written message at some node which can be read by any agent visiting that node. Thus, in our model, each node of the network is provided with a *whiteboard*, i.e., a local storage where agents can write and read (and erase) information; access to a whiteboard is done in mutual exclusion. The whiteboards are also used for marking the nodes. Initially, the homebases of the agents are marked[3]. The amount of whiteboard memory available at a node, is limited; $O(\log n)$ bits suffice for our algorithms.

Initially the agents do not know the graph or its topology. The only a-priori knowledge the agents may have is either the size of the graph, n, or the total number of agents present, k.

2.2 Problems and Constraints

The *Labelled Map Construction* problem consists in having the agents construct the same map of the graph, where both edges and nodes are labelled; the edges labels are same as those of the graph, while there is no a-priori restriction on the node labels except that each node must have a unique label. We assume that the amount of local memory available with an agent is enough to store such a map.

[2]This assumption is necessary because otherwise an agent cannot even explore a simple three-legged-star graph.

[3]Since both nodes and agents are *anonymous* this marker denotes that the node is the homebase of some agent, but cannot be used to break symmetry.

Formally, the LMC problem can be stated as follows. An instance of the problem consists of a graph $G(V, E)$ with an edge-labelling λ defined on G, and a placement function $p : V \rightarrow \{0, 1\}$ which defines the initial locations of the $k = |\{v \in V : p(v) = 1\}|$ agents. A given instance (G, λ, p) of the LMC problem is said to have been solved by a distributed algorithm $\mathcal{A}$, if on executing the algorithm $\mathcal{A}$, each agent obtains a node-labelled, edge-labelled map of the graph, (with the position of the agent marked in it) such that the label assigned to any particular node is the same in all the maps.

In the rest of this section, we look at the relationship between the LMC problem and other related problems for multiple agents dispersed in an unknown environment. For each of these, the problem instance is defined in the same manner as for the LMC problem.

- The *Labelling* problem : The problem of assigning unique labels to the nodes of an unlabeled graph. A given instance (G, λ, p) of the *Labelling* problem is said to have been solved when the whiteboard of each node is marked with a label and no two nodes have the same label.
- The *Agent Election* problem(AEP) : The problem of electing a leader among the agents. A given instance (G, λ, p) of the AEP problem is said to have been solved when exactly one of the k agents reaches the state 'LEADER' and all other agents reach the state 'FOLLOWER'.
- The *Rendezvous*(RV) problem : The problem of gathering all the agents together in one node. A given instance (G, λ, p) of the Rendezvous problem is said to have been solved when all the k agents are located in a single node of G.
- The *Spanning Tree Construction*(SPT) problem : The problem of constructing a spanning tree of the graph. A given instance (G, λ, p) of SPT problem is said to have been solved if each edge of the graph G is marked as either a Tree-edge or Non-Tree edge, such that the set of Tree-edges, T represents a spanning tree of the graph G (i.e. (V, T) is a tree).

For any of the above problems, an instance of the problem is said to be solvable if there exists a deterministic algorithm $\mathcal{A}$ such that every execution[4] of the algorithm $\mathcal{A}$ solves the particular instance of the problem.

Theorem 2.1 *The LMC problem is solvable for the instance (G, λ, p) if and only if the AEP problem is solvable for the same instance.*

Proof : *LMC => AEP*: Once the LMC problem has been solved, agent election can be done without making any extra moves. When each agent has a uniquely labelled map of the graph, the agent whose homebase has the smallest label in the map, (among all the homebase nodes), changes to 'LEADER' state and all the other agents change their state to 'FOLLOWER'.
AEP => LMC: Once a leader agent is elected, this agent can explore the graph, marking the nodes with unique labels, while the other agents remain stationary in their homebases. Thus the leader agent can construct the map and then it can visit the homebase of each agent to communicate the map to the other agents. Note that only $2(m + n)$ extra moves are required in this case. ■

It is interesting to note that solving the *Rendezvous* problem is also equivalent to solving the agent election problem, in the presence of whiteboards. The mutual exclusion property of the whiteboards allow us to break the symmetry among the agents and elect a leader, once the agents rendezvous at a single node. On the other hand, solving the LMC problem solves both *Labelling* and *Rendezvous*. Once the agents have a labelled map they can mark the whiteboard of the nodes with the respective labels. The agents can then move to the node having the smallest label, thus achieving *Rendezvous*.

[4]Recall that we are considering an asynchronous system and the agents can start execution at any arbitrary time.

Both these tasks could be done in $O(k.n)$ moves. Finally, note that solving the *Labelling* problem helps us to solve the LMC problem. Once the graph is labelled, each agent can execute a depth-first traversal of the labelled graph, to obtain a uniquely labelled map of the graph. During the depth-first traversal, each agent would make $2m$ moves, for total of $2k.m$ moves.

Thus the problems of *Labelling*, *Rendezvous*, *Agent Election* and *Labelled Map Construction* are computationally equivalent in our model. However the SPT problem is not equivalent to these four problems, in general.

The relationship among these problems can be used to determine the conditions for solvability of the LMC problem, based on previous results for the leader election problem.

Lemma 2.1 *The LMC problem is not solvable if the agents know neither the value of n (the size of the graph) nor k (the number of agents present).*

Lemma 2.2 *For k agents in a graph of size n, the LMC problem is not solvable, in general, if* $\gcd(n,k) > 1$ *i.e. if n and k are not co-prime.*

Notice that when n and k are not co-prime, it is possible that the agents are initially placed in exactly symmetrical positions with respect to each other (provided that the graph itself is symmetrical; e.g. a ring), such that, no deterministic algorithm can break the symmetry among the agents and achieve leader election. Also note that Lemma 2.1 holds even when $\gcd(n,k) = 1$.

For the rest of this paper, we shall assume that $\gcd(n,k) = 1$ and the agents have prior knowledge of the value of at least one of n and k. Under these conditions, we show that it is possible to solve the *Labelled Map Construction* problem, using just $O(\log n)$ node memory and making $O(k.m)$ moves.

3 Distributed Traversal

As a preliminary step in our solution protocol, we will have the agents perform an initial cooperative exploration of the graph. We want the agents to explore the graph collectively, in such a way that the total number of edge traversals is minimized. Each agent can traverse an area around its homebase, while avoiding the parts being explored by the other agents. During the exploration, the agent needs to remember the path to its homebase, so that it does not get lost. Each agent stores in its memory the sequence of labels (in order) of edges traversed by it, starting from the homebase. We call this sequence of labels as the *Exploration-Path* (or, simply the *Path*). When the edge $e = (u,v)$ is traversed by the agent from u to v, then the label $\lambda_v(e)$ is appended to the *Path*. This enables the agent to return back to the previously visited node (i.e. u) whenever it wants to. When it does so, the agent is said to have backtracked the edge e and the label $\lambda_v(e)$ is deleted from the *Path*. So, at all times during the traversal, the *Path* contains the sequence of labels of the links that an agent has to traverse (in reverse order) to return to its homebase from the current node.

Each agent on wake-up, starts traversing the graph, from the homebase and it marks the visited nodes if they are previously unmarked[5]. The agent also builds a partial *Map* of the territory that it marks. The algorithm executed by each agent is the following

Algorithm *EXPLORE*

```
1. Set $Path$ to empty ;
   Initialize the Territory $\mathcal{T}$ as single-node graph consisting of the homebase;
```

[5]Recall that the homebases are already marked.

```
2. While there is another unexplored edge $e$ at the current node $u$,
         mark link $\lambda_u(e)$ as T-edge and then traverse $e$ to reach node $v$;
         If $v$ is already marked,
                  return back to $u$ and re-mark the link $\lambda_u(e)$ as NT-edge;
         Otherwise
                        mark $v$ as explored and mark $\lambda_v(e)$ as T-edge;
                        Add link $\lambda_v(e)$ to $Path$;
                        Add the edge $e$ and node $v$ to territory $\mathcal{T}$;

3. When there are no more unexplored edges at the current node,
   If $Path$ is not empty then,
      remove the last link from $Path$, traverse that link and repeat Step 2;
   Otherwise, Stop and return $\mathcal{T}$;
```

We make the following observations about the effects of this algorithm.

Lemma 3.1 *During the execution of algorithm EXPLORE,*
(a) *If an agent marks a node, it eventually traverses each edge incident to it.*
(b) *Every node in the graph is marked by exactly one agent.*
(c) *The territory marked by the agent is a connected subgraph of G.*
(d) *There is no cycle consisting of only 'T' edges.*

Lemma 3.2 *The total number of edge traversals made by the agents in executing algorithm EXPLORE, is at most* $4.m$*, irrespective of the number of agents.*

Note that for this algorithm, we do not require much memory at the whiteboards of the nodes. In fact one bit per whiteboard is sufficient—for marking the node as explored.

When an agent A finishes executing algorithm EXPLORE, A would have obtained a map of the territory marked by it. Lemma 3.1 says that the territory of each agent is a tree, the territories marked by different agents are all disjoint, and together they span the whole graph. So, the distributed traversal of the graph by multiple agents creates a spanning forest of the graph. The edges belonging to the territory of some agent are marked as T-edges (i.e. tree-edge) and those edges not included in any territory as NT-edges (i.e. Non-Tree-edge). Note that the nodes at the two end-points of an NT-edge may either belong to the same tree or two different trees.

4 Merging the Maps: Spanning Tree Construction

To obtain the map of the whole graph, the maps constructed by the agents after the execution of algorithm EXPLORE, need to be merged somehow. The task of merging together the maps (i.e. the territories) of the agents, is complicated by the fact that the maps constructed by two agents may look exactly similar. Also there may be cyclic 'NT' edges connecting two nodes of the same tree. To avoid the cyclic edges, we would first construct a spanning tree of the graph by joining the trees marked by different agents.

In this section, we show how the agents can construct a spanning tree of the graph and then finally use it to obtain a complete map of the graph. Here, the reader may recall the well-known distributed algorithm for minimum spanning tree construction(MST) given by Gallager, Humblet and Spira [18], where the spanning tree is constructed by repeatedly joining adjacent trees using the unique edge of minimum weight connecting them. Such an approach is unfortunately not applicable in our setting, since neither the edges nor the nodes have unique labels, making it impossible for the agents to agree on a unique edge for joining two trees.

Thus, in our setting, we need a much more complicated protocol for merging the maps and building the spanning tree. Such a distributed algorithm, MERGE-TREE, is described in the following.

This new algorithm uses the algorithm *EXPLORE* as a procedure. The algorithm proceeds in phases, where in each phase, some agents become passive i.e. they stop participating in the algorithm. Agents communicate by writing certain symbols on the whiteboards. Two special symbols would be used which we call the 'ADD-ME' symbol and the 'DEFEATED' symbol. An agent can be in one of three states: *Active*, *Defeated* or *Passive.* Each agent is *active* at the time it starts the algorithm, but it may become *defeated* and subsequently *passive*, during some phase of the algorithm. When an agent becomes *passive* during a phase, it keeps waiting at its current location till the end of the algorithm. At the time an agent starts the algorithm, it knows the value of either n or k.

Algorithm *MERGE-TREE*

Phase 0 : Each agent on startup executes procedure EXPLORE and when it finishes, it has a map of the territory marked by it and also a count of the number of nodes marked. Each agent maintains a *Token* which is of the form (Ph, Nc, Ac) where Nc (Node-Count) is the count of the number of nodes marked by it, Ac (Agent-Count) is the number of agents in its territory (initially set to 1) and Ph is the phase number which is also initially set to 1. Now the agent can begin the first phase.

In phase i, $1 \leq i < k$, an agent A (if *active*), executes the following steps:

STEP 1 – 'WRITE-TOKEN' : Agent A does a depth-first traversal of its territory using the map; recall that a territory is a tree. During the traversal the agent writes its Token on the whiteboard[6] of each node in its tree.

STEP 2 – 'COMPARE TOKEN' : During this step, the agent compares its Token with the Tokens in adjacent trees. Agent A starts a depth-first traversal of its territory. During the traversal, whenever it finds an 'NT' edge $e = (u, v)$ incident to some node u in its territory, it traverses the edge e to reach the other end v, compares its Token with the Token at v, and takes an appropriate action, before returning back to u. If it does not find any Token at node v (or, finds a Token from the previous phase $i - 1$), it waits till the Token for phase i is written at v. On the other hand, if it finds a Token from phase $i + 1$ at node v, it ignores that Token, goes back to u and continues with the traversal.

Two Tokens from the i-th phase, $T_1 = (i, N_1, K_1)$ and $T_2 = (i, N_2, K_2)$, are compared as follows. Token T_1 is said to be larger than Token T_2 if either $N_1 > N_2$, or $N_1 = N_2$ and $K_1 > K_2$. The two tokens are said to be equal if both $N_1 = N_2$ and $K_1 = K_2$. Otherwise, Token T_1 is smaller than Token T_2.

After the comparison of Tokens, the agent takes one of the following actions:

[<] If the Token at the other side is larger, it writes a 'ADD-ME' symbol on the whiteboard of node v and returns to node u. It remembers[7] the node u, (as the *terminal* node) and the edge e (as the *bridge* edge). It then does a complete traversal of its territory writing 'DEFEATED' symbols on each node in its territory. It now becomes *defeated.* (The actions taken by a *defeated* agent are described below.)

[6] Any previously written Token or symbol is deleted from the whiteboard.

[7] The agent remembers a node by marking in its map.

[=] If the Token at the other side is equal to its own Token, it ignores the Token, returns to its own tree and continues with its traversal.

[>] If the Token at the other side is smaller, it waits at node v till it finds a 'DEFEATED' symbol. On finding a 'DEFEATED' symbol, it goes back to u and continues with the traversal.

If agent A becomes *defeated* then it takes the following actions. It continues with the traversal and Token comparisons — whenever it finds a Token which is smaller or equal to its Token, it takes the same action as an *active* agent; but, when it finds a Token that is larger than its Token, it ignores it. (So, a *defeated* agent never writes any 'ADD-ME' symbol.) After completing the traversal, the *defeated* agent A returns to the *terminal* node u and marks the *bridge* edge e as a 'T' edge. It then traverses the edge e to reach the other end, say v. It adds the edge e to its map and designates the vertex corresponding to node v, as the *junction point*, in the map. At this stage, the agent A becomes *passive* and does not participate in the algorithm anymore.

During the traversal, whenever an *active* (or *defeated*) agent A finds an 'ADD-ME' symbol at some node w in its tree, it takes the following action. It deletes the 'ADD-ME' symbol and waits at node w till the agent B (which had written the message), returns back to w. Agent A then acquires all the information available in agent B's memory, including B's Token, its map and all other Tokens and maps acquired earlier by agent B. Agent A also remembers the vertex corresponding to node w, as the location where it acquired this new information. (This vertex is called the *acquisition point.*)

STEP 3 – 'UPDATE TOKEN' : If the agent A completes the second step without becoming *passive*, it extends its territory and updates its Token, before starting the next phase. The agent adds together the Node-count and Agent-count values respectively, from all the acquired Tokens, including its own Token, to get the new values of Node-count Nc, and Agent-count Ac. The new phase number is obtained by incrementing Ph by one. The agent also constructs a new map by merging the acquired maps with its own map. Note that the agent has the information about how to merge the maps[8]. (While merging the maps, the agent may have to relabel some of the vertices of the maps, to ensure unique labelling of the vertices.) The resulting map constructed by the agent defines its new territory.

On updating the Token, if the agent finds that the new node-count is equal to n (or the agent-count is equal to k), then it reaches the termination condition. Otherwise, it proceeds with the next phase.
Phase k : An agent which reaches this phase terminates the algorithm after sending failure notification to all agents in its territory.

When an agent A reaches the termination condition, it becomes the leader agent; at this stage, it has a spanning tree of the whole graph. Finally, it executes the following procedure:

Procedure COMPLETE-MAP

1. The leader agent executes a depth-first traversal of the spanning tree, writing node labels on the appropriate whiteboards.
2. The leader agent traverses the graph, adding the non-tree edges to the map.
3. The leader agent traverses the spanning tree to communicate the full map to all the agents.

5 Analysis of the Algorithm

In this section, we show the correctness of our algorithm and analyze its complexity. We use the following notations. G_{iA} denotes the subgraph of G that corresponds to the territory of agent A at the

[8]The maps are disjoint except for the joining vertex.

time when it reaches the end of phase i. If A becomes *passive* in phase i, then $G_{iA} = \phi$. We denote by Γ_i the set of all agents which start phase i, in *active* state. We say that the algorithm reaches phase i, if there is at least one agent that starts phase i.

Whenever an agent A becomes *defeated* on comparing its Token with the Token of an agent B, during phase i, we say that agent A was defeated by agent B in phase i. In that case we know that B was *active* at the start of phase i and B's Token in phase i was larger than A's Token, in phase i.

The following facts imply that there is no deadlock in the algorithm MERGE-TREE.

Lemma 5.1 ***(a)*** *An (*active*) agent that starts phase i, either completes the phase or becomes* passive *during the phase.* ***(b)*** *At the end of every phase i reached by the algorithm MERGE-TREE, there is always at least one* active *agent.*

Proof : Part(a): We show that there cannot be any cyclic waiting among the agents. Suppose, for the sake of contradiction, that there is a group of agents $A_1, A_2, \ldots, A_t$ such that for each $1 \leq j \leq t-1$, A_j waits for A_{j+1}, and A_t waits for A_1. We represent these agents as vertices of a graph and we draw directed (colored) arcs to denote who waits for whom. (The color of the edge denotes the type of waiting.) There are three situations when an agent A, in phase $i \geq 1$ has to wait at a node v for some agent B:

1. Agent A found no Token, or a Token from phase $(i-1)$ at node v and it is waiting for the Token for phase i to be written, by agent B. [Denoted by *Blue* arc.]

2. Agent A is waiting at node v, after finding an 'ADD-ME' symbol written by B. [Denoted by *Yellow* arc.]

3. Agent A found agent B's Token (at node v) to be smaller than its own Token. This indicates A is waiting for agent B to write a 'DEFEATED' symbol at v. [Denoted by *Red* arc.]

Note that in first case above, agent B is either in a lower phase than A, or B is yet to complete step-1 of phase i. But in the other two cases, both A and B have to be in step-2 of the same phase i and also B would have a smaller Token than A in that phase. Also note that an agent can be waiting only if it is in step-2 (i.e. the COMPARE-TOKEN step) of some phase.

So, each $A_j, 1 \leq i \leq t$, is in step-2 of some phase and for any $1 \leq j, k \leq t$,

- there is a blue arc from A_j to A_k if and only if A_k is in smaller phase than A_j.

- there is a red or yellow arc from A_j to A_k if and only if A_k is in the same phase as A_j but has a smaller Token than A_j.

Let us first consider the case when at least one of the arcs in the cycle is blue. Let A_j be the first node with a blue arc in the cycle (connecting A_j to A_{j+1}). Let A_j be in phase i. By definition of red and yellow arcs, we then know that any A_k with $k < j$ are in the same phase i. If $j = t$, we immediately have a contradiction because, by definition of blue arc, A_1 would have to be in a phase smaller than i. Let us then assume that $2 \leq j \leq k-1$. Also in this case we have a contradiction because, by definition of blue arc, $A_{j+1}, A_{j+2}, \ldots, A_t$ must be in a smaller phase than phase i. So, there can neither be a blue nor a red (or yellow) arc from A_t to A_1. Thus, we cannot have a cycle containing any blue arc.

We now consider the case when the cycle is composed by yellow and red arcs only. In this case, all the agents in the cycle would be in the same phase i, and each agent would have a smaller Token than the agent on its left — which is not possible!

So, we conclude that there can not be any cyclic waiting among the agents. Each agent in phase i, either reaches the end of the phase or becomes *passive*.

Part(b): Note that an agent A can be defeated by an agent B during phase i, only if B's Token in phase i is larger than A's Token, in phase i. So, an agent A having the largest Token in phase i cannot be defeated in phase i. Thus, agent A remains *active* at the end of phase i. ■

We shall show next that the algorithm MERGE-TREE terminates in finite time, and whenever $\gcd(n,k) = 1$, there is exactly one leader agent on termination and the map constructed by the leader agent is a spanning tree of the graph G.

Lemma 5.2 *The following holds for any phase i that is reached by the algorithm:*

1. *For each $A \in \Gamma_i$, G_{iA} is a tree.*
2. *For any $A, B \in \Gamma_i$, if A and B are distinct, then $G_{iA} \cap G_{iB} = \phi$.*
3. *$H_i = \bigcup_{A \in \Gamma_i} G_{iA}$, is a subgraph of G having the same vertex set as G.*

Proof : For phase $i = 0$, the territory obtained by each agent in phase i is the same as obtained from the EXPLORE algorithm. Thus, the given conditions hold in phase $i = 0$, as proved in section 3. We assume that these conditions hold at some phase $i = r$ that is reached by the algorithm and we show that each of these conditions would continue to hold at phase $i = r + 1$, if the algorithm reaches phase $r + 1$.

For any agent A which reaches phase $r + 1$, the following holds: If agent A does not find any 'ADD-ME' symbol during phase r, then $G_{(r+1)A} = G_{rA}$. On the other hand, if agent A read an 'ADD-ME' symbol in phase $r + 1$, then it acquires the territory of some *defeated* agent B that wrote the 'ADD-ME' symbol in phase $r + 1$. The territory of such a *defeated* agent B consists of G_{rB} combined with a single edge e that connects a node in G_{rA} to a node in G_{rB} (where G_{rA} and G_{rB} are disjoint trees, by our assumption). Thus, the new territory of A at the end of phase $r + 1$ would still be a tree.

Agent A acquires some territory from agent B in phase $r + 1$, only if it defeats agent B in phase $r + 1$. Note that an agent can be defeated only once and by only one agent. Thus, if A and C are two agents that reach the end of phase $r + 1$, then both of them could not have acquired the territory of the same agent B. This implies that the territories of the agents A and C would remain disjoint at the end of phase $r + 1$. Thus, $G_{(r+1)A} \cap G_{(r+1)C} = \phi$ for any two agents A and C that reaches the end of phase $r + 1$.

Note that whenever an agent becomes *passive* in phase $r + 1$, its territory is acquired by the agent that defeated it. So, each node that was contained in the territory of some *active* agent at the end of phase r, would be contained in the territory of some *active* agent at the end of phase $r + 1$. In other words $H_{r+1} \supseteq H_r$. Thus, the third condition also holds for phase $i = r + 1$. ■

For an agent $A \in \Gamma_i$, we define $NodeCount(G_{iA})$ to be the number of nodes in G_{iA}. Note that, this is equal to the Nc part in the Token of agent A for phase $i + 1$. Similarly, $AgentCount(G_{iA})$ is defined to be the number of nodes in G_{iA} that are agent homebases. This is equal to the Ac part in the Token of agent A in phase $i + 1$. We have the following corollary as a consequence of the above lemma:

Corollary 5.1 *For any phase i reached by the algorithm, we have*

$$\sum_{A \in \Gamma_i} NodeCount(G_{iA}) = n \quad and \quad \sum_{A \in \Gamma_i} AgentCount(G_{iA}) = k$$

Two agents A and B are said to be neighbors in phase i, if G contains an edge $e = (u, v)$ such that G_{iA} contains the vertex u and G_{iB} contains the vertex v. Note that the edge e is not included in either G_{iA} or G_{iB} (as $G_{iA} \cap G_{iB} = \phi$), so e would remain to be marked as an 'NT' edge at the end of phase i.

Lemma 5.3 *If* $\gcd(n, k) = 1$ *then, for any phase* $i \geq 1$ *with* $|\Gamma_i| \geq 2$*, at least one agent* $B \in \Gamma_i$*, becomes* passive *during phase* i*.*

Proof : If $t = |\Gamma_i| \geq 2$ then t agents would have reached the end of phase $i - 1$. Note that all the t agents cannot have the same node-count and agent count at the end of phase $i - 1$ (because then we would have $\gcd(n, k) \geq t \geq 2$). So, there must be two (neighboring) agents A and B, with different Tokens and thus, one of them would defeated in the Token comparison during phase i. ■

So, if the condition $\gcd(n, k) = 1$ is satisfied, then in each phase, at least one of the *active* agents, becomes *passive*, until in some phase i, there is only a single *active* agent left. The territory of this agent A, would be the tree G_{iA} containing all the nodes of G (due to Lemma 5.2), and the node-count and agent-count of A would equal n and k respectively. Thus, agent A would reach termination condition and the algorithm would terminate. Notice that the algorithm always terminate within k phases, irrespective of the values of n and k.

Lemma 5.4 *If an agent* A *reaches phase* $i = k$*, then* $\gcd(n, k) > 1$.

Proof : Due to Lemma 5.3, if $\gcd(n, k) = 1$, only one of the k agents can reach phase $k - 1$ and in that case, the territory of this agent in phase $k - 1$ would contain all the n nodes of G and thus, this agent would terminate at the end of phase $k - 1$. ■

Hence, the algorithm fails only if $\gcd(n, k) > 1$.

Theorem 5.1 *The algorithm MERGE-TREE terminates after at most k phases, and if* $\gcd(n, k) = 1$ *then exactly one agent* A *reaches the termination condition; when this happens,* G_{iA} *represents a spanning tree of* G*.*

Theorem 5.2 *After executing the procedure COMPLETE-MAP, every agent has a uniquely labelled map of the graph.*

This proves the correctness of our algorithm.

Theorem 5.3 *The number of edge traversals made by the agents during algorithm MERGE-TREE is in* $O(k.m)$ *where* m *in the number of edges in the graph* G*.*

Proof : During procedure EXPLORE, the agents perform at most $4m$ moves. During each phase of the algorithm MERGE-TREE, each 'T' edge is traversed twice in step-1 and twice in step-2, during the depth-first traversals; Each NT edge is traversed at most four times in step-2. This accounts for $4m$ moves per phase and thus a total of $O(k.m)$ moves. Other than that each defeated agent does one extra traversal of its tree to write 'DEFEATED' messages. These extra moves would account for at most $O(k.n)$ edge traversals. Finally, the procedure COMPLETE-MAP takes $O(m)$ edge-traversals. ■

As for the memory requirement, only $O(\log n)$ bits of whiteboard memory are needed per node of the graph, which is sufficient for writing a token on any whiteboard.

6 Reducing the number of phases

When the values of n and k are co-prime, then the algorithm will never fail to elect a leader. In case the agents have the prior knowledge that n and k are co-prime, then we can simplify the algorithm and reduce its complexity by allowing only those agents which defeat some other agent to proceed to the next phase. This new algorithm is described below. As before the algorithm proceeds in phases and during the algorithm an agent can be one of the states – *Active*, *Passive*, or LEADER. Every agent begins in state *Active* with the initial knowledge of the value of both n and k (or else the value of $\gcd(n, k)$ along with one of the values n or k).

Algorithm Explore-&-Capture

If $\gcd(n, k) > 1$, terminate the algorithm with failure notification. Else proceed with the first phase. In each phase $i \geq 1$, an agent A (if *Active*) executes the following steps:

STEP 1: Agent A executes procedure EXPLORE using the phase number i as a tag for all marks that it makes. During the execution of EXPLORE, if agent A at some node v, finds a mark with phase number $j < i$ then v is considered to be unmarked. (In this case, agent A overwrites such marks with its own mark.) On the other hand, if node v is marked with $j > i$, then agent A aborts the execution of EXPLORE and changes to *Passive* state.

The territory obtained by an *active* agent A at the end of procedure EXPLORE is denoted by T_{iA}. Let n_i be the number of nodes and k_i be the number of homebases in T_{iA}. The token for agent A in this phase would be $Q_A = (i, n_i, k_i)$.

If $n_i = n$ (or equivalently $k_i = k$) then agent A changes to state LEADER and executes procedure COMPLETE-MAP. Otherwise, it continues with the next step.

STEP 2: An active agent A performs a depth-first traversal of the territory T_{iA} and writes the token Q_A on the whiteboard of each node in the territory.

STEP 3: At the start of this step, agent A initializes its *Win-Count* to zero and then starts another depth-first traversal of its territory. During the traversal, whenever it finds an outgoing edge $e = (u, v)$ (where $u \in T_{iA}$ but $v \notin T_{iA}$), agent A visits the node v and reads the token written at node v. If agent A finds no tokens from phase i or higher, at node v, then the agent waits until such a token, (say Q_B) is written and then takes the following action, before continuing with the traversal:

(i) If $(Q_B > Q_A)$ agent A writes 'ADD-ME' at the node v, traverses its territory writing 'DEFEATED' on each node in T_{iA} and then changes to passive state.

(ii) If $(Q_B < Q_A)$ then agent A waits at node v, until it finds a 'DEFEATED' symbol written at node v.

During this step, whenever agent A finds an 'ADD-ME' symbol written in any node of T_{iA}, it deletes the 'ADD-ME' symbol and increments its *Win-Count.* When agent A completes this step, if the *Win-Count* of agent A is still zero then it changes to passive state.

Any agent that becomes passive in this phase returns to its homebase and waits for a notification from the Leader agent. Those agents that did not become passive, continue with the next phase.

The procedure COMPLETE-MAP is same as before.

Lemma 6.1 *For any active agent A executing STEP-3 of some phase i of the algorithm, the following holds: (a) Agent A does not wait forever in STEP-3 (b) Agent A either itself becomes passive or causes another agent from phase i to become passive.*

Proof : Part(a): First notice that if agent A is in STEP-3 of phase i, then there is at least one other agent in phase i, and all those agents which are neighbors of agent A must have at least reached

STEP-1 of phase i. Now, suppose agent A is waiting to find a token from phase i at node v (which belongs to the territory of agent B). So, agent B (which is in phase i) must write its token at node v during Step-2 (unless it became passive in STEP-1, in which case there is an agent in higher phase which would reach and mark node v) and then agent A would stop waiting. In the other case of waiting, suppose agent A found a smaller token Q_B at node v and is waiting to find the 'DEFEATED' symbol. In that case, agent B is at least in Step-2 of phase i, so in Step-3 it will find a larger token and write 'DEFEATED' on all nodes in T_{iB}.
Part(b): If agent A did not became passive in Step-3, then it must have seen at least one 'ADD-ME' symbol. Notice that the agent that wrote this 'ADD-ME' symbol must be in Step-3 of phase i and thus it would become passive during this Step. ■

Lemma 6.2 *If* $\gcd(n, k) = 1$ *then, for any phase* $i \geq 1$ *with* $|\Gamma_i| = r \geq 2$*, the following holds: (a) At least* $r/2$ *agents become* passive *during phase i, and (b) At least one agent reaches phase* $i + 1$.

Proof : Part(a): Suppose $r' \leq r$ agents complete Step-3 of phase i without becoming passive, then each of these r' agents must have caused some other agent from this phase to become passive (due to Lemma 6.1). So, $r' \leq r/2$ and $(r - r') \geq r/2$ agents become passive in phase i.
Part(b): If $\gcd(n, k) = 1$, then among the r agents $\in \Gamma_i$, there would be two neighboring agents A and B, such that token(A) ¿ token(B). So, during Step-3 of phase i, agent B would find a larger token and thus write 'ADD-ME' symbol at some node v. The agent whose territory in phase i contains node v would reach phase $i + 1$. ■

Theorem 6.1 *Algorithm* Explore-&-Capture *terminates after at most* $\log(k)$ *phases, electing a unique leader agent and constructing a uniquely labelled map of G if and only if* $\gcd(n, k) = 1$.

Proof : If $\gcd(n, k) \neq 1$ then every agent terminates before starting the first phase. If $\gcd(n, k) = 1$, then due to Lemma 6.2, in some phase i there would be only one agent $A \in \Gamma_i$. Thus, in phase i, the territory of agent A, T_{iA} would be a spanning tree of the graph G and agent A would become leader and execute procedure COMPLETE-MAP to obtain a uniquely labelled map of G. Again due to Lemma 6.2, $k/2^{i-1} > |\Gamma_i| = 1$ which implies that $i \leq \log(k)$. ■

Theorem 6.2 *The number of edge traversals made by the agents during algorithm* Explore-&-Capture *is in* $O(m. \log(k))$ *where m in the number of edges in the graph G. The algorithm requires* $O(\log n)$ *memory at each node.*

Proof : Using arguments similar to those for the previous algorithm, it can be shown that each edge is traversed a constant number of times in each phase of algorithm *Explore-&-Capture.* Thus, there are $O(m)$ edge-traversals per phase of the algorithm, adding up to a total of $O(m \log k)$ traversals for the whole algorithm. Notice that this algorithm requires the same amount of whiteboard memory as the previous algorithm. ■

7 Conclusions

We have considered the problem of constructing a labelled map of an unknown unlabelled graph by a team of identical asynchronous mobile agents initially dispersed among the nodes of the graph. Multiple agents can collectively explore the whole graph, each obtaining a partial map of the graph. To combine the partial maps and construct a single combined map of the whole graph, there has to be some agreement between the agents. We have given an algorithm that achieves this agreement and elects one of the agents as a leader, under the necessary condition that $\gcd(n, k) = 1$, where n is the size of the graph and k is the number of agents. For our algorithm to work, it is sufficient that the value of one of n or k is known to the agents. However, if the value of $\gcd(n, k)$ is also known to the agents then it possible to have a more efficient algorithm.

The Labelled Map Construction(LMC) problem is related to the problems of *Rendezvous* of agents, *leader election*, *graph labelling* and *spanning tree construction*, such that a solution to LMC problem solves these other problems too. We have given solutions to these problems under the weakest possible model where both the nodes of the graph and the agents are anonymous, there is no synchronization among the agents and the knowledge available to the agents is the minimum that is necessary to solve the problem. Previous solutions to any of these problems are either restricted to specific topologies or are applicable in a much stronger model. For instance, in [4], the leader election problem was solved, under the same condition that $\gcd(n, k) = 1$, but assuming that the graph is endowed with *sense of direction*. Thus, our result indicates that the same result can be achieved even when there is no *sense of direction*, if the value of either n or k is known.

In this paper, we have considered only those instances of the problem where $\gcd(n, k) = 1$, because we know that LMC problem is unsolvable in general, when n and k are not co-prime. However, when $\gcd(n, k) > 1$, there are still some specific instances of the problem which are solvable. For example, if the graph is a tree with an odd number of nodes, then the LMC problem is solvable regardless of whether n and k are co-prime or not. Thus, future research on the problem should be directed towards finding an *effective* solution to the problem, i.e. an algorithm that is able to detect if the problem is solvable in a given setting and then solves it whenever it is possible to do so.

References

[1] S. Albers and M. Henzinger, "Exploring unknown environments", In Proc. 29th Annual ACM Symp. Theory Comput., 416–425, 1997.

[2] S. Alpern and S. Gal, *The Theory of Search Games and Rendezvous*, Kluwer, 2003.

[3] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro, "Can we elect if we cannot compare?", In *Proc. 15th ACM Symp. on Parallel Algorithms and Architectures* (SPAA'03), 200–209, 2003.

[4] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro, "Election and rendezvous in fully anonymous systems with sense of direction", In *Proc. 10th Coll. on Structural Information and Communication complexity* (SIROCCO'03), 17–32, 2003.

[5] M. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan, "The power of a pebble: Exploring and mapping directed graphs", In *Proc. 30th ACM Symp. on Theory of Computing* (STOC'98), 269–287, 1998.

[6] M. Bender and D. K. Slonim, "The power of team exploration: two robots can learn unlabeled directed graphs", In *Proc. 35th Symp. on Foundations of Computer Science* (FOCS'94), 75–85, 1994.

[7] S. Das, P. Flocchini, A. Nayak, and N. Santoro, "Distributed exploration of an unknown graph", In *Proc. 12th Coll. on Structural Information and Communication Complexity* (SIROCCO '05), LNCS 3499, 99–114, 2005.

[8] X. Deng and C. H. Papadimitriou, "Exploring an unknown graph". *J. of Graph Theory* 32(3), 265–297, 1999.

[9] A. Dessmark, P. Fraigniaud, and A. Pelc, "Deterministic rendezvous in graphs", In *Proc. 11th European Symposium on Algorithms (ESA'03)*, 184–195, 2003.

[10] A. Dessmark and A. Pelc, "Optimal graph exploration without good maps", In *Proc. 10th European Symposium on Algorithms (ESA'02)*, 374–386, 2002.

[11] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc, "Tree exploration with little memory", *Journal of Algorithms*, 51:38–63, 2004.

[12] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, "Robotic exploration as graph construction", *Transactions on Robotics and Automation*, 7(6):859–865, 1991.

[13] P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, C. Sawchuk, "Multiple mobile agent rendezvous in a ring". Proc. *6th Latin American Theoretical Informatics Symp.* (LATIN'04), 599–608, 2004.

[14] P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc, "Collective tree exploration", In *6th Latin American Theoretical Informatics Symp.* (LATIN'04), 141–151, 2004.

[15] P. Fraigniaud and D. Ilcinkas, "Digraph exploration with little memory", Proc. *21st Symp. on Theoretical Aspects of Computer Science* (STACS'04), Montpellier, 246–257, 2004.

[16] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg, "Graph exploration by a finite automaton", In *29th Symposium on Mathematical Foundations of Computer Science* (MFCS), 451–462, 2004.

[17] P. Fraigniaud, A. Pelc, D. Peleg, and S. Perennes, "Assigning labels in unknown anonymous networks with a leader", *Distributed Computing* 14(3), 163–183, 2001.

[18] R. G. Gallager, P. A. Humblet, and P. M. Spira, "A distributed algorithm for minimum-weight spanning trees", *ACM Transactions on Programming Languages and Systems* 5(1), 66–77, Jan. 1983.

[19] E. Korach, S. Kutten, S. Moran, "A Modular Technique for the Design of Efficient Distributed Leader Finding Algorithms", *ACM Transactions on Programming Languages and Systems* 12(1), 84–101, 1990.

[20] E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk, "Mobile agent rendezvous in a ring", In *Int. Conf. on Distibuted Computing Systems* (ICDCS 03), 592–599, 2003.

[21] P. Panaite and A. Pelc, "Exploring unknown undirected graphs", Proc. *9th ACM-SIAM Symp. on Discrete Algorithms* (SODA'98), 316–322, 1998.

[22] P. Panaite and A. Pelc, "Impact of topographic information on graph exploration efficiency", Networks, 36 (2000), 96–103, 2000.

[23] N. Sakamoto, "Comparison of initial conditions for distributed algorithms on anonymous networks", Proc. *18th ACM Symposium on Principles of Distributed Computing* (PODC'99), 173–179, 1999.

[24] X. Yu and M. Yung, "Agent rendezvous: A dynamic symmetry-breaking problem", In *Int. Coll. on Automata Languages and Programming* (ICALP'96), 610–621, 1996.