Distributed Exploration of Unlabelled Graphs by Multiple Agents*

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

Abstract

We consider a distributed version of the typical graph exploration problem where a mobile agent has to traverse the edges of an unlabelled (i.e., anonymous) graph and return to its starting point, building a map of the graph in the process. In our case, instead of a single agent, there are k identical (i.e., mutually indistinguishable) agents initially dispersed among the n nodes of the graph. The agents can communicate by writing to the *small* public bulletin boards available at each node. The objective is for each agent to build an identically labelled map of the graph; we call this the *Labelled Map Construction* problem. This problem is much more difficult than exploration by a single agent, because it involves achieving cooperation among multiple agents. In fact, this problem is deterministically unsolvable in some cases. We present deterministic algorithms that successfully solve the problem under the condition that the values of n and k are co-prime to each other. We also show how the problem of Labelled Map construction is related to other problems like leader election and rendezvous of agents.

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

1 Introduction

The problem of exploring and mapping an unknown environment has been studied extensively 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, and searching a computer network using mobile software agents. In these cases, the environment to be explored is often modelled as a graph, (or a digraph) where a single entity (called an agent or a robot) starting at one of the nodes of the graph, has to traverse through all the edges of the graph and returns back to the starting point, constructing a map of the graph in the process. The problem has been studied under two scenarios. In the first case, when the nodes of the graph are labelled with unique identifiers, then the problem is reduced to that of *Traversal*, that is of visiting every node and every link of the graph, and the issue becomes that of efficiently performing a traversal of the unknown graph. In the other case, i.e. in the absence of such unique labels, the task becomes more difficult, but can still be achieved if the agent is supplied with a marking device for marking the nodes during exploration.

In this paper, we consider the distributed version of this problem where instead of a single agent, there are now several identical agents starting at different nodes of the graph and each of the agents is trying to explore the graph. The agents are identical in all respect—they have the same capabilities

*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

and follow the same protocol—in particular, they cannot be distinguished from one-another. We do not assume the presence of any synchronization between the agents (i.e. the 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. (Each node of the graph contains a *whiteboard* where any visiting agent can read or write information in fair mutual exclusion.)

The objective for each agent is to build a map of the graph that they are traversing; By a map, we mean a node-labelled, edge-labelled representation of the graph where the starting location of the agent is specially marked. Moreover, we want the maps obtained by the agents to be consistent with one another in terms of the node labelling, i.e. the label assigned to any node should be the same in every agent's map. We call this problem *labelled map construction* (LMC). The problem is solved when each agent returns to its homebase and outputs such a map of the graph. Note that having such identically-labelled maps helps the agents to coordinate with each-other when performing any distributed task together; Hence the importance of this problem.

The motivation for considering unlabelled graphs is the fact that in many practical scenarios, the limited capabilities of the agents may not allow them to distinguish among the nodes of the graph. 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 distinguish one intersection from another visited previously. In the case of software agents traversing a network, the identification field of the nodes (i.e. the network hosts) may be intentionally 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 multiple agents can successfully map an anonymous graph.

This problem, as we show, is closely related to some others basic problems, such as *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 the related problems of *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.

We 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 $O(m \cdot k)$ 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).

The algorithms presented in this paper are generic protocols which can be executed on arbitrary graphs. In fact, we do not assume any prior knowledge of the environment except the knowledge

of the network size or the number of agents present. Our algorithms are designed for one of the computationally weakest model where both the nodes of the graph and the agents are anonymous. Further, our solutions are deterministic and include mechanisms for explicit termination detection (even for those cases which are unsolvable).

In section 2.2 we formally describe the *Labelled Map Construction* and other related problems, and discuss 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 exploration by a single agent. In labelled graphs, each node is uniquely identifiable and hence, it is always possible to explore and map the graph simply by traversing it. 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, 2, 14, 12, 25, 26, 27]). The travelling agent is sometimes modelled as finite automata and sometimes as a token.

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 [18, 15]). For exploring arbitrary anonymous graphs, various methods of marking nodes have been used by different authors. Bender *et al.* [9] 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.* [16] used a set of distinct markers to explore unlabeled undirected graphs. Yet another approach, used by Bender and Slonim [10] 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 used earlier by Fraigniaud and Ilcinkas [19] for exploring directed graphs and by Fraigniaud et al. [18] for exploring trees. In [15, 19] the authors focus on minimizing the amount of memory used by the agents for exploration (however, they 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 [10], whereas Fraigniaud et al. [18] 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.

Solutions to the leader election problem based on underlying exploration algorithms were given by Korach et al.[23] and by Afek et al.[1]. Both these papers use travelling tokens spawned by the nodes of the network, which are similar to the mobile agents considered here. However, these solutions were for labelled networks. Another more classical solution for the leader election problem in arbitrary but labelled networks was given by Gallager et al.[22]. For anonymous networks, the problem of electing a leader was studied first by Angluin [4] and later by Yamashita and Kameda [30] and independently by Boldi and Vigna [5], finally leading to a complete characterization of those graphs(digraphs) where the problem is deterministically solvable. Sakamoto [28] studied the effect of initial conditions for performing distributed computations in anonymous networks.

In the mobile agent model, another problem which has been studied extensively is that of gathering all the agents at a single node, called the *Rendezvous* problem (see [3] for a recent survey). However, most of the results obtained for this problem use probabilistic algorithms. Among deterministic solu-

tions to the problem, the investigations of Yu and Yung on synchronous graphs of known topology [31] and of Dessmark *et al.* on synchronous rings and graphs [13] 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 [17, 24], and those of Barriere *et al.* on graphs with *sense of direction* [8]. The results obtained in [8] 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. In [7], Barriere *et al.* consider solutions to the agent election problem in presence of distinct but incomparable agent labels.

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 to that node[1]. 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 to 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, $l_u : \{(u, v) \in E : v \in V\} \rightarrow \{1, 2, 3, \ldots, d(u)\}$ defines the labelling on its incident edges. We use Δ to denote 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 and can move from a node to any of the neighboring nodes in G. The private memory available to an agent is assumed to be large enough to store a complete map of G. An agent at a node u can choose to leave through any incident edge $e = (u, v)$ simply by specifying the label $l_u(u, v)$ of the edge. On reaching the node v, the agent knows the label $l_v(v, u)$ of the edge through which it arrived. It also knows the number of agents present at node v and can communicate directly with the other agents present in that node, if any.

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 restricted by fair mutual exclusion. The whiteboards are also used for marking the nodes. Initially, the homebases of the agents are marked[2]. Note that the amount of whiteboard memory available at a node would be, in general much smaller compared to the private memory of an agent; in fact only $O(\log n)$ bits of whiteboard memory suffice for our algorithms.

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

[2] 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.

Initially, the agents do not know the graph or its topology. The minimum initial knowledge available to each agent is either the size of the graph, n, or the total number of agents present, k (or both). It is possible that different agents start with different initial knowledge (e.g. some with the value of n and some with that of k).

2.2 Problems and Constraints

We now show the relationship between the LMC problem and some other well-known problems under the same setting. For each of the problems that we consider, we represent an instance of the problem with the tuple (G, λ, p) where G is a connected undirected graph, λ is a local orientation on G, and $p : V(G) \rightarrow \{0, 1\}$ is a bi-coloring on the nodes of G. A *solution* to a given problem instance is a deterministic algorithm such that when each agent locally executes the algorithm, on *termination* of the algorithm, the states of the agents and the contents of the whiteboard satisfy certain conditions (these are specific to the particular problem as described below). An agent terminates its local execution when it reaches one of the designated *final* states at which point it is not required to perform any further computation. We say that the algorithm has *terminated* when every agent has locally terminated its execution. The knowledge that the algorithm has terminated (i.e. knowledge of global termination) may not be available to each individual agent. In other words, we do not require *global termination detection* for the problems that we consider here. We only require each agent to terminate its local execution and be aware of the fact it has terminated. (This can be called *local termination detection*.)

We now define each problem in terms of the conditions to be satisfied on termination of any proposed solution algorithm for the problem.

- The *Labelled Map Construction* problem : A given instance (G, λ, p) of the LMC problem is said to have been solved when each agent obtains a *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.

- The *Labelling* problem (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* (AEP) problem (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 is in the final state 'LEADER' and all other agents are in the final state 'FOLLOWER'.

- The *Rendezvous*(RV) problem (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 (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.

For any of the above problems, an instance (G, λ, p) of the problem is said to be solvable if there exists a deterministic algorithm $\mathcal{A}$ such that every execution[3] of algorithm $\mathcal{A}$ on that particular instance, succeeds in solving the problem. We now discuss the conditions for solvability of the above problems.

[3]Recall that we are considering an asynchronous system where different possible execution sequences can give different results.

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 solved too, 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, assigning (i.e., writing) unique labels to the nodes, 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 allows 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*. Both these tasks can be performed in $O(k \cdot n)$ 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 makes $2m$ moves, for a total of $2k \cdot m$ moves.

Hence, 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 (see [8]).

Lemma 2.1 *There exists no algorithm that can solve any arbitrary solvable instance of the LMC problem, without the knowledge of neither the value of* n *(the size of the graph) nor* k *(the number of agents present).*

Lemma 2.2 *For* k *agents dispersed in a graph of size* n*, the LMC problem is not solvable in arbitrary graphs, unless* $\gcd(n, k) = 1$ *(i.e.,* n *and* k *are 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. On the other hand, if n and k are co-prime, we can always solve the AEP or LMC problems, irrespective of the graph topology or the initial placement of agents. (The use of primality for symmetry breaking in distributed systems is quite common, see [6] for example.) However, notice that the conditions of Lemma 2.1, namely the prior knowledge of n or k, is necessary even when we restrict ourselves to those instances of the problem where $\gcd(n, k) = 1$.

In the following, we present algorithms for solving the LMC problem when the agents have prior knowledge of the value of at least one of n and k. Our algorithms are effective for all those instances of the problem where $\gcd(n, k) = 1$.

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 the *Exploration-Path* (or, simply the *Path*). When an edge $e = (u, v)$ is traversed by the agent from u to v, 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 its homebase and marks each of the visited nodes that are previously unmarked[4]. The agent also builds a partial *Map* of the territory that it marks. Those edges included in the territory $\mathcal{T}$, are marked as 'T'-edge and those not included are marked as 'NT'-edge. 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;

2. While there is another unexplored edge e at the current node u,
   mark link $\lambda_u(e)$ as a '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 a 'NT'-edge;
   Otherwise
   mark v as explored and mark $\lambda_v(e)$ as a 'T'-edge;
   Add link $\lambda_v(e)$ to *Path*;
   Add edge e and node v to the 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 $\mathcal{T}$;

The above algorithm is a distributed version of the standard depth-first-search algorithm [29]; The only difference here is that since there are multiple anonymous agents, an agent does not distinguish between nodes visited by itself and nodes visited by other agents. When k agents, starting at distinct nodes of the graph G, independently execute the above algorithm, the following results can be observed. First note that the execution of each agent terminates in finite time, because every edge that is traversed, is marked and G is finite.

Lemma 3.1 *When every agent has terminated its execution of algorithm EXPLORE, the following holds:*

[4]Recall that the homebases are already marked.

(a) *If some node u was marked by an agent A, then each edge incident to u was traversed by agent A.*

(b) *Every node in the graph is marked by exactly one agent.*

(c) *The territory marked by any agent is a connected subgraph of G.*

(d) *There are no cycles in G consisting of only 'T'-edges.*

Proof: Part(a): During algorithm EXPLORE, if an agent A marks a node u, then each incident link at u is either marked by the agent A, or remains unmarked. (Thus, even if an edge $e = (u, v)$ is traversed from v to u by another agent B, the link $l_u(e)$ at u could be marked only by agent A.) Whenever there are unexplored (i.e. unmarked) links at the current node u, agent A traverses one of the unexplored edges and whenever an agent leaves a node u through an unexplored link, it eventually returns to u at some time during the traversal. Thus, each link incident to u is traversed by A before it stops. ■

Part(b): From part(a), we know that an agent visits all neighbors of a node that it marks. Also, we know that whenever an agent visits a node v, if v is unmarked, the agent marks it. To begin with, an agent's homebase is already marked and whenever a node is marked all its neighbors are eventually marked. Thus, each node would be marked by some agent. Note that due to the mutual exclusion property of whiteboards, a node cannot be marked by more than one agent. ■

Part(c): This follows immediately from the algorithm.

Part(d): If an edge $e = (u, v)$ was marked as 'T'-edge, then both nodes u and v were marked by the same agent. This means that 'T'-edges marked by distinct agent are never incident on the same node. So, if there is a cycle consisting of 'T'-edges, all these edges were marked by a single agent A. Suppose $e = (u, v)$ was last edge to be marked in this cycle, then node v was unexplored when A reached it and thus, all the edges incident at v were unmarked; This contradicts the assumption that e was the last edge to be marked. ■

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

Proof: During the execution of algorithm EXPLORE, each 'T'-edge is traversed twice—once in the forward direction and once while backtracking. Each 'NT'-edge is traversed four times, twice from each side. As there are $(n - k)$ 'T'-edges, the total number of moves (i.e. edge traversals) made by the agents is $(4m - 2n + 2k)$. ■

When an agent A finishes executing algorithm EXPLORE, A has 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 be exactly identical. In addition, some of the 'NT'-edges may be connecting two nodes of the same

tree. While merging the maps, we want to avoid such cyclic edges; Thus, 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 [22], where the spanning tree is constructed by repeatedly joining adjacent trees using the unique edge of minimum weight connecting them. Such an approach, unfortunately, is 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.

We present below a distributed algorithm called MERGE-TREE, for solving the LMC problem using procedure EXPLORE as a preliminary step. 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, including two special symbols 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 it receives a termination notification from some other agent. At the start of the algorithm, every agent knows the value of k. (We show later knowledge of the value of n instead of k would also suffice.)

Algorithm *MERGE-TREE*

Phase 0 : An agent A on startup executes procedure EXPLORE to obtain a map of its territory and a count of the number of nodes marked. The agent constructs 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. In case, $k = 1$ (or equivalently, $Nc = n$), then agent A can terminate the algorithm by executing procedure COMPLETE-MAP. Otherwise the agent begins 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 own territory using the map; recall that a territory is a tree. During the traversal the agent writes its Token on the whiteboard[5] 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 $K_1 = K_2$.
After the comparison of Tokens, agent A 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[6] node u as the *terminal* node and edge e as the *bridge*

[5] Any previously written Token or symbol is deleted from the whiteboard.
[6] The agent remembers a node by marking in its map.

edge. Agent A then performs 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.)

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

[>] If the Token at the other side is smaller, agent A 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 own Token, agent A takes the same action as an *active* agent; but, when it finds a Token that is larger than its own Token, agent A ignores this Token. (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 edge e to reach the other end, say v. It adds 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 a *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 wrote 'ADD-ME') 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 agent A completes the second step without becoming *passive*, it extends its territory and updates its Token as follows. Agent A 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. Agent A 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[7]. (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 agent-count is equal to k (or equivalently, the node-count is equal to n), 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. We shall show that in this case, $\gcd(n, k) > 1$ (See Lemma 5.4)

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.

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

Remark: In the above algorithm, the knowledge of either n or k is used to determine when an agent has reached termination condition (i.e., when the algorithm succeeded). However, for determining the failure condition (i.e. phase number $= k$), the agents need to know the value of k. In case k is not known and n is known instead, then the agents would use the following equivalent condition. If an agent's node-count Nc does not change for r consecutive phases where $r \geq n/Nc$ then, the agent can assume that it has reached phase k and consequently it can terminate the algorithm after failure notification. We shall show later (see Lemma 5.5) that this assumption is always correct.

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 time when it reaches the end of phase i (after UPDATE-TOKEN). 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 both A and B were *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 are no deadlocks in 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 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 which agent 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. (This is denoted by a *Blue* arc.)

2. Agent A is waiting at node v, after finding an 'ADD-ME' symbol written by B. (This is denoted by a *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. (This is denoted by a *Red* arc.)

Note that in first case above, agent B is either in a lower phase than A, or agent B is yet to complete step-1 of phase i. However, in the other two cases, both A and B are in step-2 of the same phase i and B has 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 $A_j, 1 \leq j \leq t$, is in step-2 of some phase and for any $1 \leq x, y \leq t$,

- there is a *Blue* arc from A_x to A_y if and only if A_y is in smaller phase than A_x.

- there is a red or *Yellow* arc from A_x to A_y if and only if A_y is in the same phase as A_x but has a smaller Token than A_x.

Let us first consider the case when at least one of the arcs in the cycle is blue. Let A_x be the first node with a blue arc in the cycle (connecting A_x to A_{x+1}). Let A_x be in phase i. By definition of red and yellow arcs, we then know that any A_y with $y < x$ are in the same phase i. If $x = t$, 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 $2 \leq x \leq t-1$. In this case we have a contradiction too, because by definition of blue arc, $A_{x+1}, A_{x+2}, \ldots, A_t$ 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 of yellow and red arcs only. In this case, all the agents in the cycle are in the same phase i, and each agent has 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. ■

Next, we show 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 that 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 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 did not find any 'ADD-ME' symbol during phase r, then $G_{(r+1)A} = G_{rA}$. On the other hand, if agent A reads 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 $r+1$ is still 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 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}. Notice 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 vertex u and G_{iB} contains vertex v. Note that edge e is not included in either G_{iA} or G_{iB} (as $G_{iA} \cap G_{iB} = \phi$), so e remains 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 are active at 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 we would have $\gcd(n,k) \geq t \geq 2$, by Lemma 5.2). So, there must be two (neighboring) agents A and B, with different Tokens and thus, one of them would be 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 (by Lemma 5.1). 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 the 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 k, then* $\gcd(n,k) > 1$ *and the territory of every active agent is of the same size.*

Proof: Due to Lemma 5.3, if $\gcd(n,k) = 1$, then at least one agent is defeated in each phase and thus, at most one of the k agents can reach phase $k-1$ and no agent can reach phase k. On the other hand, if some agent reaches phase k, then there was some phase $i < k$ during which no agent was defeated, i.e., every active agent had the same token in phase i. Thus, in all subsequent phases, the node-count of the agents cannot increase. ■

Lemma 5.5 *If an agent A has the same territory (of size z) for $r > 2$ consecutive phases (say, from phase i to $i+r$), where $r \geq n/z$, then* $\gcd(n,k) = 1$ *and agent A has completed at most $2k$ phases of the algorithm.*

Proof: If agent A has the same territory in phase i and $i+1$, then all the neighboring agents of A have the same token. If agent A has the same token for r phases, then there are at least $r-1$ other active agents having the same token and thus, the same node-count z. However, since $r \cdot z \geq n$ and

the territories of active agents are disjoint, there can be at most r active agents in phase i. Thus there are exactly r active agents in phase i, each having the same node-count $z = n/r$ and the same agent-count k/r. Hence $\gcd(n, k) > 1$. Also, note that both i and r are less than k, so $i + r$ is at most k. ■

From Lemma 5.4 and Lemma 5.5, it follows that algorithm MERGE-TREE fails only if $\gcd(n, k) > 1$. Combining all the above lemmas, we have the following results:

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.*

These theorems prove the correctness of our algorithm.

Theorem 5.3 *The number of edge traversals made by the agents during algorithm MERGE-TREE is* $O(k \cdot m)$*, 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 $O(k \cdot m)$ moves. Other than that, each defeated agent performs one extra traversal of its tree to write 'DEFEATED' messages. These extra moves would account for $O(k \cdot n)$ edge traversals. Finally, 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

The algorithm from the previous section always succeeds in electing a unique leader, whenever the values of n and k are co-prime. However, if $\gcd(n, k) > 1$, it is not always possible break the symmetry between the agents. In this case, the agents must detect this and terminate the algorithm to avoid a possible deadlock (or live-lock) situation. Now, suppose the agents have the prior knowledge that $\gcd(n, k) = 1$, then we can simplify the algorithm and reduce its complexity by allowing only those agents which defeat some other agent during some phase i to proceed to phase $i + 1$. This revised algorithm is described below. As before, the algorithm proceeds in phases and during the execution an agent can be in one of the states – *Active*, *Passive*, or *Leader*. Every agent begins in state *Active*. For this algorithm, we assume that each agent has the initial knowledge of the value of $\gcd(n, k)$ and the value of at least one of n or k. (If the agents know both n and k that is sufficient, because they can compute $\gcd(n, k)$ in that case.)

Algorithm Explore-&-Capture

If $\gcd(n, k) > 1$, then terminate the algorithm after 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 marking the nodes as *explored*. During the execution of EXPLORE, if agent A finds some node v marked with a

phase number $j < i$ then v is considered to be unmarked. In this case, agent A overwrites such marks with its own mark.

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

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 its territory G_{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 'NT'-edge $e = (u, v)$ incident to the current node u, agent A traverses e to visit node v and reads the whiteboard at node v. Agent A waits at node v until it finds a token from phase i, say Q_B (or an *explored* mark from a higher phase), and then agent A takes the following action, before continuing with the traversal:

- If $(Q_B > Q_A)$ agent A writes 'ADD-ME[i]' at the node v, traverses its territory writing 'DEFEATED[i]' on each node in G_{iA} and then changes to passive state.
- If $(Q_B < Q_A)$ then agent A waits at node v, until it finds a 'DEFEATED[i]' symbol written at node v.
- If there is an *explored*(j) mark from a phase $j > i$, then agent A aborts this step, traverses its territory writing 'DEFEATED[i]' on each node in G_{iA} and then changes to passive state.

(Note that if $Q_B = Q_A$, then agent A immediately returns to node u and continues with the traversal.)

During this step, whenever agent A finds an 'ADD-ME[i]' symbol written in any node of G_{iA}, it deletes the 'ADD-ME[i]' 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 until it receives the map from the *Leader* agent. Those agents which did not become passive, continue with the next phase. Finally, when an agent becomes *Leader*, it executes procedure COMPLETE-MAP to terminate the algorithm and provide a copy of the map to every other agent.

In the following, we show the correctness of this algorithm. We define Γ_i similarly as before to denote the set of agents that reach phase $i \geq 1$. We say that agent A *defeated* agent B (or agent B was defeated by agent A) in phase i, if agent B wrote 'ADD-ME[i]' on the whiteboard of some node $\in G_{iA}$. We also give a more precise definition of what is meant by *neighboring* agents. At any instant during the execution of the algorithm, two (*active*) agents A and B are said to be neighbors if there exists an NT-edge $e = (u, v) \in G$, such that u is marked by agent A and v is marked by agent B. (Here, A and B may not be distinct agents, i.e. an agent can be its own neighbor.)

Lemma 6.1 *The following holds at any time during the execution of algorithm Explore-&-Capture.*

(a) If an agent A is in phase $i > 1$ then each of its neighbors must be in phase $i - 1$ or higher.

(b) If an agent A is in step-3 of phase i then each of its neighbors must be in phase i or higher.

Proof: Part(a): If agent A has completed phase $i - 1$, then during step-3 of that phase, it must have visited the territory of each neighbor and waited for the token for phase $i - 1$ to be written. This means that each neighbor has reached phase $i - 1$.
Part(b): Suppose agent A is in Step-3 of phase i and one of its neighbor B is in phase $j < i$. In that case, during Step-1 of phase i agent A must have overwritten the nodes in B's territory with its own mark. So, agents A and B cannot be neighbors—a contradiction! ■

Observation 6.1 *For any phase $i \geq 1$ reached by the algorithm Explore-&-Capture, Lemma 5.2 and Corollary 5.1 still hold.*

This means that the territories of the agents in any phase i are disjoint and they span all the nodes of the graph G. Notice that the territories in phase i are defined by a fresh execution of EXPLORE, that is independent of the previous executions of procedure EXPLORE (due to the use of phase numbers). Thus, the above observation follows from the results proved for algorithm EXPLORE in Section 3.

Lemma 6.2 *For any phase $i \geq 1$ with $|\Gamma_i| = r \geq 2$, the following holds: (a) At least one agent reaches phase $i+1$. (b) At most $r/2$ agents can reach phase $i+1$.*

Proof: Part(a): If the algorithm reaches phase $i \geq 1$ then the condition $\gcd(n,k) = 1$ holds and thus, there must be two neighboring agents A and B with distinct tokens in phase i. (This follows from the above observations and the fact that the graph G is connected.) Thus the agent having the smaller token among A and B would be defeated when it finds a larger token during step-3. Since an agent can be defeated only by another agent from the same phase having a strictly larger token, it follows that there must be, in phase i, a sequence of agents $A_1, A_2, \ldots, A_t$ such that for $1 \leq r < t$, agent A_r was defeated by agent A_{r+1} and A_t remained undefeated. So, unless agent A_t became passive on encountering an explored(j) mark from a higher phase (in which case we already have an agent which reached phase i+1), agent A_t would have Win-Count > 0 at the end of step-3 and thus it would reach phase $(i+1)$.
Part(b): If an agent $A \in \Gamma_i$ reaches phase $(i+1)$, then it must have defeated some other agent in phase i. Clearly, an agent can be defeated by at most one agent. This means that for every agent $A \in \Gamma_i$ which reaches phase $(i+1)$, there is another agent $B \in \Gamma_i$ that became passive in phase i. Hence the result follows. ■

Theorem 6.1 *Assuming that the agents have prior knowledge of both n and k, algorithm* Explore-&-Capture *terminates after at most $\log(k)$ phases, solving the LMC problem if and only if $\gcd(n,k) = 1$.*

Proof: In case $\gcd(n,k) \neq 1$ then every agent terminates before starting the first phase. Otherwise $\gcd(n,k) = 1$, then due to Lemma 6.2, there exists a phase i such that there is only one agent $A \in \Gamma_i$. Thus, in phase i, the territory of agent A, T_{iA} is a spanning tree of the graph G and agent A becomes the leader and executes procedure COMPLETE-MAP to obtain a uniquely labelled map of G. In this case, due to Lemma 6.2(b), $k/2^{i-1} > |\Gamma_i| = 1$ which implies that $i \leq \log k$. Hence, the algorithm terminates in at most $\log k$ phases. ■

Theorem 6.2 *The number of edge traversals made by the agents during algorithm* Explore-&-Capture *is $O(m.\log(k))$ where m is the number of edges in 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 and there are at most $\log k$ phases which adds 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, for writing the *Token*. ■

7 Conclusions

In this paper, we considered the problem of constructing a labelled map of an arbitrary and unlabelled graph G by a team of identical asynchronous mobile agents initially dispersed among the nodes of the graph. Our solution works in two stages—first, the agents independently explore and mark the graph, each obtaining a partial map of the graph; In the second stage, the partial maps are combined to construct a map of the whole graph. Our algorithm solves the Labelled Map construction problem, while also performing leader election among the agents and constructing a spanning tree of the graph, under the assumption that the size of the graph, n is co-prime to the number of agents, k. Under this assumption, the problem is always deterministically solvable irrespective of the structure of the graph G or the initial placement of the agents on it. (Notice that the same cannot be said for the case when $\gcd(n, k) > 1$.)

An important property of our solutions is that the agents explicitly terminate their execution even in the unsolvable cases. For termination detection, the value of at least one of n or k must be known to the agents. We show that when the value of both these parameters are known, it is possible to terminate earlier, thus improving the complexity of the solution.

The problem of Labelled Map Construction(LMC) is related to several other well-studied problems such as *leader election*, *rendezvous*, *graph labelling* and *spanning tree construction*, and any solution to the LMC problem solves these other problems too. Previous studies on these problems have typically avoided the difficult scenario where the system is both anonymous and asynchronous. There have been some solutions which work for specific topologies such as ring networks (as in [17, 24]). Another solution given in [8], assumes the presence of specific labelling on the graph providing a *sense of direction* to the agents. Our solutions can be seen as an extension to these results because we have made much weaker assumptions about the model, leading to more generic algorithms. Further extensions to this work should focus on solving the problem regardless of whether n and k are co-prime or not. It is interesting to note that there are certain specific instances of the problem which are solvable even if $\gcd(n, k) > 1$, e.g., when the given graph is a tree with an odd number of nodes. However, the protocols presented in this paper do not take such specific instances into consideration.

References

[1] Y. Afek and E. Gafni, "Distributed Algorithms for Unidirectional Networks", *SIAM Journal on Computing*, 23(6):1152–1178, 1994. (Preliminary version appears in Proc. of PODC 1984).

[2] S. Albers and M. Henzinger, "Exploring unknown environments", In *Proc. 29th ACM Symp. on Theory of Computing* (STOC '97), 416–425, 1997.

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

[4] D. Angluin, "Local and global properties in networks of processors", In *Proc. 12th ACM Symp. on Theory of Computing* (STOC '80), 82–93, 1980.

[5] P. Boldi and S. Vigna. An effective characterization of computability in anonymous networks. In *Proc. 15th Int. Conference on Distributed Computing (DISC'01)*, 33–47, 2001.

[6] J. Burns and J. Pachl, "Uniform stabilizing rings," *ACM Transactions on Programming Languages and Systems*, 11(2): 330-344, 1989.

[7] 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.

[8] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro, "Election and rendezvous in fully anonymous networks with sense of direction", *Theory of Computing Systems*, 2006 (to appear). Preliminary version in *Proc. 10th Coll. on Structural Information and Communication Complexity* (SIROCCO '03), 17–32, 2003.

[9] 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.

[10] 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.

[11] 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.

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

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

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

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

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

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

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

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

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

[21] 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.

[22] 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, 1983.

[23] 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.

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

[25] Shay Kutten, "Stepwise construction of an efficient distributed traversing algorithm for general strongly connected directed networks or: Traversing one way streets with no map", In *Proc. 9th Int. Conf. on Computer Communication* (ICCC'88), 446–452, 1988.

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

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

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

[29] R. E. Tarjan, "Depth-First Search and Linear Graph Algorithms", *SIAM Journal on Computing*, 1(2):146–160, 1972.

[30] M. Yamashita and T. Kameda, "Computing on anonymous networks: Parts I and II", *IEEE Trans. Parallel and Distributed Systems*, 7(1):69–96, 1996.

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