Effective Elections for Anonymous Mobile Agents

Shantanu Das[1], Paola Flocchini[1], Amiya Nayak[1], and Nicola Santoro[2]

[1] School of Information Technology and Engineering, University of Ottawa, Canada, {shantdas,flocchin,anayak}@site.uottawa.ca,
[2] School of Computer Science, Carleton University, Canada, santoro@scs.carleton.ca

Abstract. We present distributed protocols for electing a leader among k mobile agents that are dispersed among the n nodes of a graph. While previous solutions for the agent election problem were restricted to specific topologies or under specific conditions, the protocols presented in this paper face the problem in the most general case, i.e. for an arbitrary topology where the nodes of the graph may not be distinctly labelled and the agents might be all identical (and thus indistinguishable from each other). In such cases, the agent election problem is often difficult, and sometimes impossible to solve using deterministic means. We have designed protocols for solving the problem that—unlike previous solutions—are *effective*, meaning that they always succeed in electing a leader under any given setting if at all it is possible, and otherwise detect the fact that election is impossible in that setting. We present several election protocols, all effective. Starting with the straightforward solution, that requires an exponential amount of edge-traversals by the agents, we describe significantly more efficient algorithms; in the latter the total number of edge-traversals made by the agents is always polynomial, their difference is in the amount of bits of storage they required at the nodes.

1 Introduction

1.1 The Framework

We consider the problem of leader election in distributed networked environments that support autonomous mobile agents. More specifically, there are k identical mobile agents dispersed among the n nodes of a network (or simply an undirected graph) and the agents can autonomously move from node to neighboring node throughout the network. Communication between agents is done using public whiteboards available at the nodes. An agent can communicate with another by leaving a written message at some node, which can be read by any agent visiting that node. The objective is to elect one of the agents as the leader.

We are interested in systems where both the networks and the agents are *anonymous*. The reason for this interest is because these systems provide the (computationally) weakest environments; thus, they provide insights on the nature and amount of computational power necessary for the solvability of tasks. Furthermore, any solution protocol so designed would work without relaying upon (and thus not requiring) the availability of name servers in the system.

First observe that, in these systems, the election problem is not always solvable using deterministic algorithms. Thus, an important line of research is the investigation of under what conditions election is indeed possible, if at all, in such systems. A connected line of research is to design efficient protocols for electing a leader under specific conditions. A third related area of research is the one of designing *effective* solution protocols; that is, protocols that in each setting within finite time will elect a leader, if election is at all possible in that setting, otherwise report that the problem is not solvable in that setting. The focus of this paper is on the latter area.

The problem of leader election among mobile agents communicating by whiteboards, has been earlier studied by Barrière *et al.* [5] and Das *et al.* [9]. Both these papers solve the problem under the constraint that n (the size of the graph) and k (the number of agents) are co-prime to each-other. These solutions are therefore, not *effective* according to our definition, since there are many settings where election is possible even when n and k are not co-prime. In this paper, we aim to improve upon these solutions by designing *effective* protocols for leader election in the mobile agent model. We are also concerned about the efficiency of the proposed solutions. The main cost measure of an algorithm in this model is the number of moves performed by the agents during the execution; another measure is the minimum size of the whiteboards required for the execution of the algorithm.

1.2 Our Results

In this paper we first extend the characterization of the conditions for election in anonymous message-passing systems as given by Yamashita and Kameda[17], to the mobile agent model. Based on this characterization, we describe a straightforward technique for *effective* election (protocol *Compare-View* described in Section 2.2), showing that effective protocols are indeed possible.

We then present a more efficient yet effective solution (algorithm *Agent-Elect* described in Section 3), that achieves leader election using only $O(m \cdot k)$ agent moves for k agents in a graph with m edges. In Section 4, we give two improvements to this algorithm which bring down its memory usage at the cost of a slight increase in the total agent moves. Table 1 below shows the comparison between the various algorithms based on the two cost measures. (Here Δ indicates the maximum degree of the graph.) In comparison with these algorithms, the non-*effective* solutions presented in [5] and in [9] have cost of $O(k \cdot n)$ and $O(k \cdot m)$ edge-traversals respectively and require $O(\log n)$ bits of node memory.

Algorithm	**Cost**		**Assumption**
	Edge-traversals	Node-Storage	
Agent-Elect	$O(k\ m)$	$O(m \log n)$	one of n or k is known
Agent-Elect-2	$O(k\ m^2)$	$O(\log n)$	
Agent-Elect-3	$O(k\ n\ m^2)$	$O(\log \Delta)$	
Compare-View	$O(k\ \Delta^{2n})$	$O(1)$	n is known

Table 1. Comparison of the cost (moves vs storage) for the proposed election algorithms

1.3 Related Work

In the message-passing distributed computing model, the characterization of conditions for computability in general and election in particular has been object of extensive investigations, starting from the pioneering work of Dana Angluin [2]. A complete characterization has been eventually provided by Yamashita and Kameda [17], and later refined by [8, 15]. The present work is based on the concept of the *view* of a node, introduced in [17]. Norris [15] improved on some of these results while Boldi *et al.* [8] gave a similar characterization for directed graphs using the notion of *fibration*. Many other authors have focussed their investigations on the issue of computability in anonymous networks having specific topologies, noticeably rings [3], hypercubes [14], and tori [6]. For a recent survey see [13].

The leader election problem is also related to the problem of spanning tree construction in a graph. Many distributed algorithms have been proposed for minimum spanning tree construction in labelled graphs (i.e. where nodes are labelled with distinct identifiers), notably the one proposed by Gallager, Humblet and Spira [11]. For anonymous networks, Sakamoto[16] gave an algorithm that builds a spanning forest of the graph under a variety of initial conditions. Korach, Kutten and Moran [12] showed that the complexity of solving leader election in any arbitrary graph depends on how efficiently the graph can be traversed.

The traversal or exploration of anonymous graphs have also been studied extensively, using different models for marking the nodes (e.g. [7, 10]). Another related problem in the mobile agent setting—that of gathering the agents (called the *Rendezvous* problem)—has been studied mainly for agents having distinct labels [1].

The agent election problem has been specifically studied by Barrière *et al.* [5] and Das *et al.* [9] but, as mentioned earlier, these solutions are not *effective.* Barrière *et al.* [4] have studied the agent election problem for networks where nodes are anonymous and edge/agent labels are distinct but incomparable.

2 Solving the Agent Election Problem

The Model: The network is modelled as an undirected graph $G(V, E)$, where the ordering or numbering on the nodes in V is unknown to the agents, i.e. the nodes are anonymous. At each node of the graph, the edges incident to it are locally labelled, so that an agent arriving at a node can distinguish among them. The edge labelling of the graph G is given by $\lambda = \{\lambda_v : v \in V\}$, where for each vertex v of degree d, $\lambda_v : \{e(v,u) : u \in V \text{ and } e \in E\} \rightarrow \{1, 2, ...d\}$ is a bijection specifying the local labelling at v.

Each agent is initially located in a distinct node of the graph, called its homebase. For simplicity, we assume that no two agents have the same homebase. Those nodes which are homebases are initially marked. Thus, the initial placement of agents in the graph G is denoted by the bi-coloring $b : V \rightarrow \{0, 1\}$ on the set of vertices, where those vertices that are colored 1(or, black) are the homebases of the agents. The agents communicate by reading and writing information on public whiteboards locally available at the nodes of the network. Access to the whiteboard is restricted by fair mutual exclusion. The agents are identical (i.e. they do not have distinct names or labels which can be used to distinguish among them) and each agent executes the same protocol. They are *asynchronous*, in the sense that every action an agent performs (computing, moving, etc.) takes a finite but otherwise unpredictable amount of time. We define the leader election problem in mobile agent systems as follows:
The Problem: The agent election problem for k agents located in the network (G, λ, b) is said to have been *solved* when exactly one of the k agents reaches the final state 'LEADER', and all other agents reach the final state 'FOLLOWER'.
Solvable Instance: A given instance (G, λ, b) of the AEP problem is said to be *solvable*, if there exists a deterministic (distributed) algorithm $\mathcal{A}$ such that every execution of the algorithm $\mathcal{A}$ on that instance, solves the problem within some finite time.
Effective Algorithm: A deterministic agent election algorithm $\mathcal{A}$ is said to be *effective* if every execution of $\mathcal{A}$ on every instance of the problem detects whether the instance is solvable, and terminates in finite time, succeeding in solving the problem if and only if the given instance is solvable.

2.1 Characterization: Conditions for solvability

In this paper we shall assume that the agents have prior knowledge of the value of at least one of the parameters n or k (which is a necessary condition as shown in [5]). Yamashita and Kameda [17] have determined the necessary and sufficient conditions for solution to the leader election problem (assuming that n is known) in the classical message passing network model. In that model, they introduced the concept of the *view* of a node v in a graph G, which is simply the infinite rooted tree with edge labels, that contains all (infinite) walks starting at v. In our model, we extend the concept of view to bi-colored views, where the vertices[3] in the view are colored black or white depending on whether or not they represent the homebase of some agent. The view of an agent is taken to be the bi-colored view of its homebase.

Definition 1. *The bi-colored view $T_v(G, \lambda, b)$ of node v, in the network (G, λ, b), is an infinite edge-labelled rooted tree T, whose root represents the node v and for each neighboring node u_i of v, there is a vertex x_i in T(with same color as u_i) and an edge from the root to x_i with the same labels as the edge from v to u_i in G. The subtree of T rooted at x_i is again the bi-colored view $T_{u_i}(G, \lambda, b)$ of the node u_i.*

The view from node u, truncated to a depth of h is denoted by T_u^h. An interesting observation is that the view T_u^h from node u, contains the view T_v^{h-d} for all such nodes v that are a distance of $d < h$ from node u. Thus, the view up to depth $2n$ from any node contains as sub-views, the views up to depth n, of all other nodes.

[3] A note on terminology: We use the word 'vertex' to refer to vertices in the view T, whereas the vertices of the original graph G are referred to as 'nodes'. Multiple vertices in the view may correspond to a single node in the graph G.

Property 1 ([15, 17]). The views $T_u(G,\lambda,b)$ and $T_v(G,\lambda,b)$ of any two nodes u and v in a graph G of size n, are equal if and only if the truncated view up to depth $n-1$, of these two nodes are equal, i.e. $T_u(G,\lambda,b) = T_v(G,\lambda,b)$ if and only if $T_u^{n-1}(G,\lambda,b) = T_v^{n-1}(G,\lambda,b)$.

The following result is known about the solvability of leader election in the message-passing network model.

Property 2 ([17]). The Leader Election Problem in a message-passing network represented by graph G having edge labelling λ, is solvable if and only if the view of a node is unique.

In [4], it was proved that there exists a simple transformation for converting a mobile agent based algorithm to distributed message passing algorithm, provided that the homebases are marked (i.e. the graph is bi-colored). This implies the following result:

Theorem 1. *The Agent Election Problem is solvable for a graph G with edge labelling λ, and bi-coloring b, if and only if the bi-colored view of an agent is unique.*

2.2 An Effective Election Protocol

When the agents know the value of n, we can use the following protocol for electing a leader (based on the approach of [17]). Each agent can traverse the graph to compute its view up to a depth of $2n-1$ and thus obtain the views (up to depth n-1) of all other agents. Since there exists a total order on views (of the same depth), it is possible to order the agents based on their views and thus solve the leader election problem. The following algorithm implements such a strategy:

```
ALGORITHM Compare-View

Set $h$ to $2n-1$;
Call Construct-View($h$) to get the bi-colored view $T_u^h$ of the homebase $u$;
Set $i$ to 1; $A[i] \leftarrow T_u^{n-1}$;
For each vertex $v$ in $T_u^h$, which is at level less than $n$,
    If $T_v^{n-1} \neq A[j]$ for any $1 \leq j \leq i$.
        then $i \leftarrow i+1$; $A[i] \leftarrow T_v^{n-1}$;
If $i \neq n$, then terminate with failure;
$A_b \leftarrow \{T_x^{n-1} \in A$ : x is a black node $\}$ ; Sort the array $A_b$ based on a fixed total order;
If $A_b[1] = T_u^{n-1}$, then become LEADER;
Else become FOLLOWER;
```

```
PROCEDURE Construct-View($h$)

(To construct the view up to level $h$, for current node $u$)
Add the current node $u$ to $T_u^h$,
If $h = 0$, return $T_u^h$;
Else,
    For i=1 to degree($u$),
        traverse link $i$ to reach node $v_i$;
        add the traversed edge and its labels to $T_u^h$;
        compute $T_{v_i}^{h-1}$ = Construct-View($h-1$), and add it to $T_u^h$;
    Return $T_u^h$;
```

Theorem 2. *The algorithm* Compare-View *is an* effective *election algorithm.*

Theorem 3. *The total number of agent moves in an execution of algorithm* Compare-View *is $O(k \cdot \Delta^{2n})$ for a graph of size n with maximum degree Δ, and k agents. The amount of node memory required by the algorithm is constant.*

3 Polynomial Solutions to the Agent Election Problem

3.1 Partial-Views

In the algorithm *Compare-View* from the previous section, each agent computes its view by traversing the complete graph which—in the absence of any topological information—takes an exponential number of moves. We want to reduce the number of the moves made by an agent to solve the problem. The approach we use for achieving that is to restrict the movements of each agent to a limited region around its homebase. Each agent would traverse only a subgraph of the original graph G and then exchange information with the other agents to obtain knowledge about the other parts of the graph. In other words, we partition the graph G into disjoint subgraphs, each *owned* by a single agent and called the agent's territory. Each agent executes the algorithm EXPLORE given below, to obtain its territory.

Algorithm *EXPLORE*:

```
1. Set $Path$ to empty ;
   Mark the homebase as explored and include it in the Territory $\mathcal{T}$.

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 (or $v$ is a homebase),
                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 $\mathcal{T}$;
```

After the agents execute algorithm EXPLORE, each agent has its own territory $\mathcal{T}$, which forms a tree consisting of the nodes marked by the agent and the 'T'-edges joining them. It can be easily shown that the territories of the agents are disjoint from each-other and together they form a spanning forest of the graph, containing exactly k trees each rooted at the homebase of some agent.

Lemma 1. *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.*

During algorithm EXPLORE, each agent A can label the nodes in its territory $\mathcal{T}_A$ by marking them with numeric identifiers, i.e. numbering them (in the order that they are visited). Thus, within the territory of an agent, each node could be uniquely identified. However, the nodes on two different trees may have the same label. So, once an agent traverses an 'NT'-edge to reach another marked node, it is generally not possible for the agent to determine if that node belongs to its own territory or that of some other agent. This fact complicates the design of our solution protocol.

Based on the territory of an agent, we define the *Partial-View* of an agent A having territory $\mathcal{T}_A$, as the finite rooted tree, such that: (i) The root corresponds to the homebase v_0 of agent A. (ii) For every other node v_i in $\mathcal{T}_A$, there is a vertex x_i in PV_A. (iii) For each edge (v_i, v_j) in $\mathcal{T}_A$, there is a edge (x_i, x_j) in PV_A. (iv) For each outgoing edge $e = (v_i, u_i)$ such that $v_i \in \mathcal{T}_A$ but $e \notin \mathcal{T}_A$, PV_A contains an extra vertex y_i (called an external vertex) and an edge $\hat{e} = (x_i, y_i)$ that joins x_i to it. (v) Each edge in PV_A is marked with two labels, which are same as those of the corresponding edge in G. (vi) Each vertex in PV_A is colored black or white depending on whether the corresponding node in G is a homebase or not. (vii) Each vertex is also labelled with the numeric identifier assigned to the corresponding node of G.

During the execution of algorithm EXPLORE, each agent can build its Partial-View. We have the following important result:

Lemma 2. *If the agent election problem is solvable for an instance (G, λ, b) then, after the execution of algorithm EXPLORE on (G, λ, b), there would be at least two agents having distinct Partial-Views.*

The above result implies that we can use the Partial-Views to distinguish between some of the agents. We fix a particular encoding for the partial-views so that we can compare among the PV's of the agents. We show below how such comparisons can be used for an effective agent election algorithm.

3.2 Algorithm *Agent-Elect*

The following algorithm proceeds in phases, where in each phase, agents compete with each other to become the leader. An agent can be in one of the following states: *Active*, *Passive*, *Leader*, *Follower* or *Fail*. Each agent starts the algorithm in *active* state and knows the value of k at start[4]. The (encoded) Partial View of an agent A in phase i is denoted PV_{iA} and Agent-Count(A) denotes the number of homebases in the current territory of agent A.

```
ALGORITHM Agent-Elect
Execute EXPLORE to construct the territory $T_A$;
$PV_{0A} \leftarrow$ COMPUTE-PV($T_A$) ;
For phase $i = 1$ to $k$ {
    If (AgentCount($A$) = $k$ ) {
        State ← Leader ;
        SEND-ALL("Success"); Exit();
    }
    SEND-ALL($PV_{iA}, i$);
    $S \leftarrow$ RECEIVE-PV ($i$);
    State ← COMPARE-PV($PV_{iA}$, $S$);
    If (State = Passive) {
        SEND-MERGE($i$);
        SEND-ALL("Defeated", $i$);
        Return to homebase and execute WAIT();
    }Else {
        RECEIVE-MERGE($i$);
        execute UPDATE-PV() and continue;
    }
}
SEND-ALL("Failure"); Exit();
```

– **Procedure** SEND-ALL(M): During this procedure, agent A simply writes the message M on the whiteboard of each node in its territory.
– **Procedure** RECEIVE-PV(i): During this procedure, agent A visits each vertex u in its territory and traverses each NT-edge $e = (u, v)$ incident at u. On reaching the node v at the other end of the edge e, agent A waits till it finds the pair (PV_{iX}, i) written at node v (where PV_{iX} is an encoded Partial-View). Each Partial-View PV_{iX} that is read is added to the set S, which is returned at the end.

[4] With a simple modification the same algorithm can be executed if the value of n is known instead of k.

- **Procedure** COMPARE-PV(PV_{iA}, S): During this procedure, agent A compares its Partial-View PV_{iA} with those in the set S. If it finds any $PV_{iX} > PV_{iA}$, agent A changes its State to *Passive*. Else, for every PV_{iY} that is less than PV_{iA}, agent A stores the corresponding node v (where it was found) to the *Defeated-List*. The procedure returns the current state of the agent (Active or Passive).
- **Procedure** SEND-MERGE(i): During this procedure, the agent A returns to the node v where it found some Partial-View PV_{iX} that is greater than its own Partial-View PV_{iA}. On reaching node v, it writes (MERGE,i,$\lambda_v(e)$) on the whiteboard of node v, where e is the NT-edge joining v to T_A.
- **Procedure** RECEIVE-MERGE(i): During this procedure, agent A visits every node v in the *Defeated-List* and waits till it finds ("Defeated",i) on the whiteboard at node v. Finally agent A visits every node u in its territory and if it finds (MERGE,i,l) written at u, then the edge e having $\lambda_u(e) = l$ is re-marked as a T-edge (from both sides). In this case, we say that the territories at the two ends of edge e are merged.
- **Procedure** UPDATE-PV(): During this procedure an active agent updates its Partial-View and its territory as follows. For every edge e that it re-marked as T-edge during this phase, it finds the corresponding edge in its Partial-View and replaces the external node v incident on this edge with the Partial-View PV_{iX} that it read at node v. The new Partial-View obtained at the end of this procedure is called $PV_{(i+1)A}$ and the internal nodes in this partial view represents the new territory of agent A.
- **Procedure** WAIT(): This procedure is executed by a Passive agent A. The agent A simply waits at its homebase until it finds either "Success" or "Failure" written on the whiteboard. If it finds "Success", then State is changed to *Follower*; otherwise the State is changed to *Fail*, and the agent terminates the algorithm.

We have the following results showing the correctness of the above algorithm.

Definition 2. *(a) Γ_i denotes the set of agents that reach phase i of the algorithm in active state. (b) We say that the algorithm reaches phase i if at least one agent reaches phase i in active state.*

Lemma 3. *(a) During any phase i of the algorithm, the territories of the agents $\in \Gamma_i$ form a spanning forest of the graph G where each territory is a connected component of G. (b)For any phase i, if $|\Gamma_i| \geq 2$, and none of the agents in $Gamma_i$ become passive during phase i, then the AEP problem is unsolvable for the given instance (G, λ, b). (c)If at least one agent becomes passive in phase i then at least one agent reaches phase (i+1) in active state.*

Theorem 4. *Given any instance (G, λ, b) of the AEP, algorithm AGENT-ELECT succeeds in electing a unique leader agent whenever (G, λ, b) is a solvable instance, and otherwise terminates with failure notification.*

Theorem 5. *The algorithm AGENT-ELECT requires $O(m \cdot k)$ agent moves, in total, for any given instance (G, λ, b) where m is the number of edges in G and k is the number of agents.*

Theorem 6. *The algorithm AGENT-ELECT requires $O(m \log n)$ memory at the nodes of the graph.*

We have the following lower bound on the cost of an effective election algorithm for mobile agents.

Lemma 4. *Any deterministic algorithm for effective leader election among k anonymous agents, dispersed in an arbitrary anonymous graph $G(V, E)$ with $|V| = n$ nodes and $|E| = m$ edges, would require $\Omega(k \cdot n)$ edge traversals, irrespective of the amount of memory available at the nodes.*

Thus from Theorem 5 and Lemma 4, we can say that the algorithm AGENT-ELECT is almost optimal in terms of agent moves, at least for sparse graphs (where $m \simeq n$).

4 Reducing the Size of the Whiteboards

Even though the number of agent moves used by algorithm AGENT-ELECT is quite efficient in terms of the number of agent moves; the memory requirements for the algorithm is much larger than that of algorithm *Compare-View* which uses constant memory. We would like to reduce the amount of memory required, without making an exponential number of agent moves. In the following we propose some modifications to our algorithm, in order to reduce the amount of memory required at the whiteboards of the nodes. As a result the number of agent moves made by the algorithm is slightly increased, even though it is still polynomial in the size of the graph.

4.1 Algorithm Agent-Elect-2

We propose the algorithm *Agent-Elect-2* which is a modified version of the previous algorithm (AGENT-ELECT) and works as follows:

```
Algorithm Agent-Elect-2
Execute EXPLORE-2 to construct the territory $T_A$;
$PV_{0A}$ ← COMPUTE-PV($T_A$) ;
For phase $i = 1$ to $k$ {
    If (AgentCount($A$) = $k$ ) {
       State ← Leader ;
       SEND-ALL("Success"); Exit();
    }
    SEND-ALL("Begin", $i$);
    $S$ ← RECEIVE-PV-2 ($i$);
    State ← COMPARE-PV($PV_{iA}$, $S$);
    If (State = Passive) {
       SEND-MERGE($i$);
       SEND-ALL("Defeated", $i$);
       Return to homebase and execute WAIT();
    }Else {
       RECEIVE-MERGE-2($i$);
       execute UPDATE-PV() and continue;
    }
}
SEND-ALL("Failure"); Exit();
```

– The procedure EXPLORE-2() is similar to procedure EXPLORE with the only difference that instead of marking the edges as T-edge and NT-edge, at each node v a parent-link would be stored which would point to the parent of node v in the territory tree.

- The procedure RECEIVE-PV-2() is different from RECEIVE-PV() in the following way. When an agent A executing RECEIVE-PV-2() reaches an external node v to read the Partial-View, it traverses the territory T_B that contains v (using the Parent-links), and computes the Partial-View PV_{iB}.
- The procedure RECEIVE-MERGE-2() is again similar to RECEIVE-MERGE() with the following changes. Whenever an agent A has to merge its territory T_A with the territory T_B at other end of an external edge (u,v), agent A sets the Parent-link at node v to point to node u and then updates (i.e. reverses) the Parent-Link for each edge on the path from v to the root of T_B.

Lemma 5. *The output of procedure RECEIVE-PV-2 is identical to the output of procedure RECEIVE-PV.*

The only difference between the two algorithms is that in the modified algorithm, an agent computes the Partial-View of its neighboring agents, instead of reading it from the whiteboard, as in the original algorithm AGENT-ELECT. Thus, the correctness of the algorithm *Agent-Elect-2* follows from the correctness of AGENT-ELECT, due to the above lemma.

Theorem 7. *The algorithm* Agent-Elect-2 *performs* $O(k \cdot m^2)$ *agent moves in total and requires* $O(\log n)$ *bits of node memory.*

4.2 Algorithm Agent-Elect-3

In order to further reduce the memory requirement of our algorithm, we can make the following modifications to algorithm *Agent-Elect-2*:

1. The procedure EXPLORE-2 is replaced by procedure EXPLORE-3, with the change that EXPLORE-3 would not explicitly write the labels of the nodes on the whiteboard of the nodes.
2. The procedure RECEIVE-PV-2() would be replaced by procedure RECEIVE-PV-3() where an agent A that is computing the Partial-View of a neighbor B and needs to read the label of a node x *external* to T_B, computes the label[5] by traversing the tree containing x.
3. During the execution of the algorithm, whenever an agent A has to write the phase number i on the whiteboard, it will write ($i \mod 3$) instead.

This new algorithm is called *Agent-Elect-3*.

Theorem 8. *The algorithm* Agent-Elect-3 *performs* $O(k \cdot n \cdot m^2)$ *agent moves in total and requires* $O(\log \Delta)$ *bits of node memory.*

Acknowledgements

The authors would like to thank Masafumi Yamashita for the many helpful discussions.

[5] The label assigned to a node v belonging to tree $\mathcal{T}$, is uniquely determined as the rank of the path to v from the root of $\mathcal{T}$, when compared with the paths to other nodes belonging to the same tree $\mathcal{T}$.

References

1. S. Alpern and S. Gal. *The Theory of Search Games and Rendezvous.* Kluwer, 2003.
2. D. Angluin. Local and global properties in networks of processors. In *Proc. 12th ACM Symp. on Theory of Computing* (STOC '80), 82–93, 1980.
3. H. Attiya, M. Snir, and M.K. Warmuth. Computing on an anonymous ring. *Journal of ACM*, 35(4), 845–875, 1988.
4. L. Barrière, 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.
5. 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.
6. P.W. Beame and H.L. Bodlaender. Distributed computing on transitive grids: The torus. In *Proc. Symp. Theor. Aspects of Computer Science* (STACS '89), 294–303, 1989.
7. 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.
8. P. Boldi, S. Shammah, S. Vigna, B. Codenotti, P. Gemmell and J. Simon. Symmetry breaking in anonymous networks: Characterizations. In *Proc. 4th Israel Symp. on Theory of Computing and Systems*, 16–26, 1996.
9. S. Das, P. Flocchini, S. Kutten, A. Nayak, and N. Santoro. Map construction of unknown graphs by multiple agents. *Theo. Comp. Sci.* (Submitted). Preliminary version in *Proc. 12th Coll. on Structural Information and Communication Complexity* (SIROCCO '05), 99–114, 2005.
10. P. Fraigniaud and D. Ilcinkas. Digraph exploration with little memory. In *Proc. 21st Symp. on Theoretical Aspects of Computer Science* (STACS '04), 246–257, 2004.
11. 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.
12. 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.
13. E. Kranakis. Symmetry and computability in anonymous networks: A brief survey. In *Proc. 3rd Int. Conf. on Structural Information and Communication Complexity* (SIROCCO'97), 1–16, 1997.
14. E. Kranakis and D. Krizanc. Distributed computing on anonymous hypercube networks. *Journal of Algorithms*, 23(1), 32–50, 1997.
15. N. Norris. Universal covers of graphs: Isomorphism to depth $n-1$ implies isomorphism to all depths. *Discrete Applied Mathematics*, 56(1), 61–74, 1995.
16. 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.
17. M. Yamashita and T. Kameda. Computing on anonymous networks: Parts I and II. *IEEE Trans. on Parallel and Distributed Systems*, 7(1), 69–96, 1996.