Computer Science
Technical Report

Colorado State University

A Self-Stabilizing Distributed Algorithm to Construct An Arbitrary Spanning Tree of a Connected Graph*

Gheorghe Antonoiu and Pradip K Srimani
Department of Computer Science
Colorado State University
Ft. Collins, CO 80523

Technical Report CS-95-106

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873

Phone: (970) 491-5792 Fax: (970) 491-2466
WWW: http://www.cs.colostate.edu

*Address for Correspondence: Pradip K Srimani, Department of Computer Science, Colorado State University, Ft. Collins, CO 80523, Tel: (303) 491-7097, Fax: (303) 491-2466, Email: srimani@CS.ColoState.Edu

A Self-Stabilizing Distributed Algorithm to Construct An Arbitrary Spanning Tree of a Connected Graph*

Gheorghe Antonoiu and Pradip K Srimani
Department of Computer Science
Colorado State University
Ft. Collins, CO 80523

Abstract

We propose a simple self-stabilizing distributed algorithm that maintains an arbitrary spanning tree in a connected graph. In proving the correctness of the algorithm we develop a new technique without using a bounded function (which is customary for proving correctness of self-stabilizing algorithms); the new approach is simple and can be potentially applied to proving correctness of other self-stabilizing algorithms.

1 Introduction

In a distributed system the computing elements or nodes exchange information only by message passing. Every node has a set of local variables whose contents specify the local state of the node. The state of the entire system, called the *global state*, is the union of the local states of all the nodes in the system. Each node is allowed to have only a partial view of the global state, and this depends on the connectivity of the system and the propagation delay of different messages. Yet, the objective in a distributed system is to arrive at a desirable global final state (legitimate state). One of the goals of a distributed system is that the system should function correctly in spite of intermittent faults. In other words, the global state of the system should ideally remain in the legitimate state. Often, malfunctions or perturbations bring the system to some illegitimate state, and it is desirable that the system be automatically brought back to the legitimate state without the interference of an external agent. Systems that reach the legitimate state starting from any illegitimate state in a finite number of steps are called self-stabilizing systems [Dij74, Dij86]. This kind of property is highly desirable for any distributed system, since without having a global memory global synchronization is achieved in finite time and thus the system can correct itself automatically from spurious perturbation or failures. Few such algorithms have recently appeared in the literature [GH90, SS92, BGW89, FD92, ADG92]; a good survey of self-stabilizing algorithms can be found in [Sch93].

Every node in a self-stabilizing system has a set of rules, each rule having two parts - an antecedent (boolean condition) part and an action part. A node is said to be *privileged* if the antecedent part of some rule is true for that node. Privileged nodes execute their action part (make a move), which involves changing the local state of the node by changing the values of the local variables. Any privileged node that makes a move is called an *active node*.

We model a distributed system by an undirected connected graph $G = (V, E)$, where V denotes the set of nodes representing the machines (processors) and E denotes the set of edges representing the interconnections among the processors. Let $|V| = n$ = the number of nodes in the graph. Our objective in this paper is twofold: (1) to develop a distributed algorithm that always maintains an *arbitrary spanning tree* of the graph, and (2) to develop a completely different graph theoretic proof technique to show the correctness of self-stabilizing algorithms. Maintaining a spanning tree in an interconnection graph is essential for most of the applications that run on a message passing distributed system.

*Address for Correspondence: Pradip K Srimani, Department of Computer Science, Colorado State University, Ft. Collins, CO 80523, Tel: (303) 491-7097, Fax: (303) 491-2466, Email: srimani@CS.ColoState.Edu

It is to be noted that there exist self-stabilizing algorithms for the spanning tree problem [HC92, SS92]; but both of these algorithms always construct a breadth-first spanning tree; they cannot recognize an arbitrary spanning tree. Also, the proof technique used in [HC92] is complicated using bounded monotonically decreasing functions defined on global system states. Most existing self-stabilizing algorithms are proved to be correct by defining a bounded function that is shown to decrease monotonically at every step [Kes88]. The proposed algorithm can recognize any arbitrary spanning tree and our proof technique is completely new and does not need any such bounded function (we have also not used any operational arguments). Another interesting feature of our algorithm is that we allow multiple privileged nodes to be concurrently active. Most self-stabilizing algorithms assume that there is a *central daemon* [Dij74] that decides which of the privileged nodes makes a move. In other words, the central daemon serializes the moves made by the privileged nodes, but the order in which the privileged nodes are chosen to make their moves is not known a priori. However, the presence of such a daemon is against the fundamental idea of a distributed system. Our algorithm does not need this assumption. When multiple nodes are privileged, an arbitrary subset of these nodes (at least one) takes action (this is defined as one move or one step in the algorithm). Since the actions taken by a node depend on the local states of the neighboring nodes, we assume that current states of the neighboring nodes dictate the action even when the neighboring nodes are active concurrently (in this sense the actions taken by nodes are still *atomic*). Our algorithm does not assume any order or any specific rule in which the set of active nodes is chosen at any point of time. This is also true for the algorithm in [HC92]. But unlike [HC92], we do not need any extra complication in the proof; our proof technique inherently accommodates concurrent actions by nodes.

2 The Algorithm

Let G be the given connected graph (V, E) where a specific node r is designated to be the root node. We use the following notations:

- *n:* number of nodes in the graph
- $\mathcal{N}(i)$*:* set of the neighbors of node i
- *L(i):* level of node i
- *P(i):* predecessor pointer of node i, pointing to one of the nodes in V.

Thus each node i maintains two data structures $L(i)$ and $P(i)$ which can have arbitrary values, i.e., $0 \leq L(i) \leq n$ and $1 \leq P(i) \leq n$ (we assume nodes are arbitrarily numbered from 1 through n). We do not need to consider level values beyond that (even after perturbation) as we can always assume each processor is capable of doing a modulo n operation and always keeps the remainder (mod n) as its level value. In the legitimate state we want a spanning tree of the connected graph G rooted at the given node r. Consider the following predicate for any node i:

$$\Psi_i = ((P(i) \in \mathcal{N}(i)) \wedge (L(i) = L(P(i)) + 1))$$

This predicate is true when the predecessor of the node i is one of its neighbors and the level of the node i is 1 greater than the level of its predecessor. Now we can define our legitimate state.

Definition 1 *The system is in a legitimate (stable) state iff* $L(r) = 0 \wedge P(r) = r \wedge \forall i \neq r : \Psi_i$.

Note: For any arbitrary valid spanning tree of the graph G, there exists a legitimate state such that the predecessor pointers correspond to the spanning tree. The converse is also true and will be proved later.

The purpose of our algorithm is to bring back the stable state once the system is in any possible illegitimate state by any perturbation. The basic idea is whenever the system is in an illegitimate state at least one of the nodes should be able to recognize it and should take some corrective action. Our algorithm has a single uniform rule for all the nodes in the graph. Each

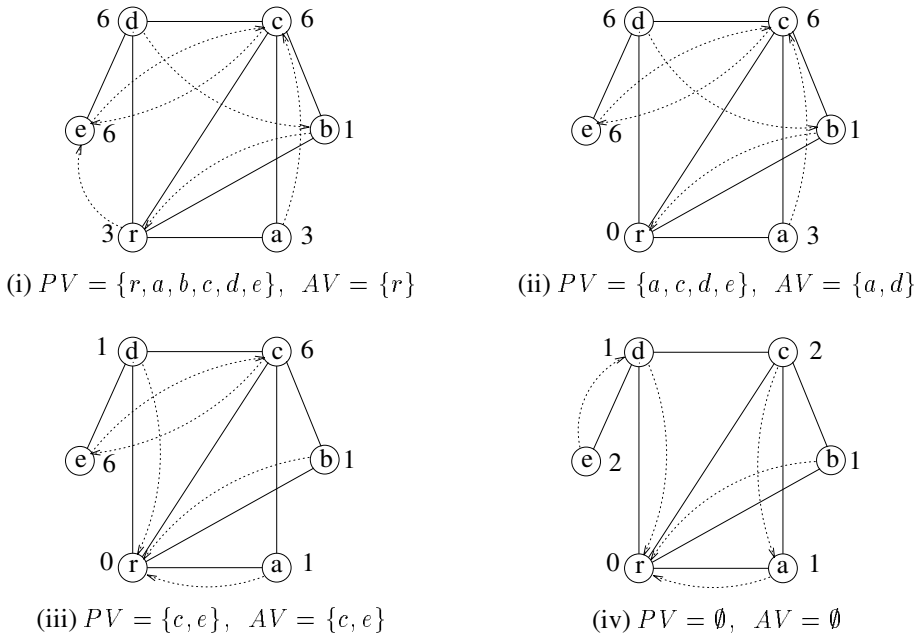

Figure 1: Stepwise Execution of the Algorithm

node looks at its own state and the states of its neighbors and takes action by changing its own level and/or its predecessor pointer. We introduce another predicate Φ_i for any node i;

$$\Phi_i = (\exists j)(j \in \mathcal{N}(i) \wedge L(j) < L(i))$$

Φ_i is true for a node i iff there exists at least one neighbor of node i with a level less than that of node i. We can now state the algorithm as a single rule for each node in the graph. The rule at node i is as follows:

$$(\mathbf{R}) \quad \begin{cases} \textbf{if } i = r \wedge (P(i) \neq r \vee (L(i) \neq 0)) \textbf{ then } P(i) = r; L(i) = 0 \\ \textbf{else if } \neg\Psi_i \wedge (L(i) < n) \wedge \neg\Phi_i \textbf{ then } L(i) = L(i) + 1 \\ \textbf{else if } \neg\Psi_i \wedge \Phi_i \textbf{ then } P(i) = j; L(i) = L(j) + 1 \end{cases}$$

Note: If there exists more than one node j for Ψ_i to be true, any one is chosen at random; the choice doesn't affect the correctness of the algorithm.

Remark 1 *The root node may be privileged in an illegitimate state, but once it takes action it becomes un privileged and can never be privileged again. This is not true for other nodes.*

Definition 2 *If a privileged node changes only its level value (and not the predecessor), we call this a type I move; otherwise if it changes both its level and predecessor we call it a type II move.*

Remark 2 *If a privileged node makes a type II move, it becomes un privileged and remains so until the new predecessor node takes some action; if a privileged node makes a type I move, it may still be privileged after the move.*

Before we prove the correctness of our algorithm, we illustrate the execution of the algorithm by using an example graph.
Example: Figure 1 illustrates the execution of the algorithm on an example graph from an arbitrary initial state. The connected symmetric graph in our example has 6 vertices, $\{r, a, b, c, d, e\}$, where r is the specified root node. Each node in the figures is labeled with its name and its level. The predecessor pointer at each node is shown by a dotted line with an arrow. The set PV

denotes the set of privileged nodes and the set AV denotes the set of active nodes. It is to be noted that in a given state more than one node may be privileged; we have arbitrarily chosen a subset to be active in our example. Figure 1(i) is the initial state and the Figure 1(iv) is the final legitimate state. It is to be noted that there are many other possible sequence of moves that will bring the state back to a stable state starting from the same initial illegitimate state.

3 Correctness Proof

In order to prove that the correctness of a self-stabilizing algorithm, we need to show that it meets the following three requirements of a self-stabilizing algorithm [Dij86]: (a) In any illegitimate state there is at least one privileged node; (b) In a legitimate state no node is privileged; and (c) For all possible initial states and for all possible ways a privileged node is selected to take action during execution of the algorithm, the system is guaranteed to reach a legitimate state after a finite number of moves.

Remark 3 *The root node r is privileged iff $(P(r) \neq r \vee L(r) \neq 0)$; any other node i ($i \neq r$) is privileged iff $\neg\Psi_i \wedge (\Phi_i \vee (\neg\Phi_i \wedge L(i) < n))$.*

Remark 4 *If a node i ($i \neq r$) is not privileged then the predicate $\Psi_i \vee (\neg\Phi_i \wedge L(i) = n)$ is true.*

Lemma 1 *In a legitimate or a stable state, no node is privileged.*

Proof : Obvious from definitions 1 and 3. □

Lemma 2 *If no node is privileged, each node in the graph has a level less than n.*

Proof : We prove by contradiction. Let $S(k) = \{i | L(i) = k\}$ be the set of nodes with level k, $0 \leq k \leq n$. Assume $|S(n)| \geq 1$. Since the root node r is not privileged, $L(r) = 0$ and hence $r \notin S(n)$. Since the graph is connected, at least one node $i \in S(n)$ must have a neighbor outside of $S(n)$. Since this neighbor has a level less than n, Ψ_i is true ($\neg\Phi_i$ is false) (Remark 4) and hence there exists a node j such that $P(i) = j$ and $L(i) = L(j) + 1$. That is, $L(j) = n - 1$ and hence, $|S(n-1)| \geq 1$. For any node $k \in S(n-1)$, Ψ_k must be true since node k has a level less than n and is not privileged. Proceeding as before, $|S(n-2)| \geq 1$. By repeating the argument we see that $|S(k)| \geq 1$ for all k, $0 \leq k \leq n$. Since these $S(k)$'s are mutually disjoint, union of all these sets has at least $(n+1)$ nodes which is a contradiction. Thus, $S(n)$ must be null, i.e., each node in the graph has a level less than n. □

Lemma 3 *If no node is privileged, the global system state represents a valid spanning tree of the graph rooted at node r.*

Proof : Since node r is not privileged, $P(r) = r$ and $L(r) = 0$. By Lemma 2, each node i ($i \neq r$) has a level less than n and hence by Remark 4 the predicate Ψ_i is true for node i. The predecessor pointers define a subgraph of G whose edges are $(i, P(i))$, $i \neq r$; this subgraph is acyclic since Ψ_i is true for each node i and is connected since each node is connected to the root. Hence, the global system state represents a valid spanning tree of the graph rooted at node r. □

Next, we prove that the algorithm brings the system back to a stable state starting from any arbitrary illegitimate state. First, we note that the local state of a node i is defined by the pair $(L(i), P(i))$ and the *global system state* is defined by the union of the local states of the nodes. Since the levels and predecessors can assume only finitely many different values, the system state space is finite. We need some definitions.

Definition 3 *For any system state, we define a spanning_tree set STS recursively as follows: (i) root node r is in STS; and (ii) any other node i is in STS iff Ψ_i is true and $P(i) \in STS$.*

Definition 4 *The set STS is called initial or ISTS if the root node is privileged and final or FSTS if the root node $r \in STS$ is not privileged.*

Example: Consider the previous example. In the initial system state (Figure 1), the $ISTS$ is $\{r\}$ since the root node r is privileged; in Figure 1(ii) the $FSTS$ is $\{r, b\}$, in Figure 1(iii) the $FSTS$ is $\{r, b, a, d\}$ and finally in Figure 1(iv) the $FSTS$ is $\{r, a, b, c, d, e\}$.
We make the following immediate observations:

- For each possible system state, there is a well defined STS (either ISTS or FSTS); the set is never null (it always contains the root node).
- No node in an ISTS is privileged except the root node r which is privileged.
- No node in FSTS is privileged.
- For any legitimate state, the set STS is an FSTS (since the root must not be privileged); for an illegitimate state, the set STS is either an ISTS or a FSTS.
- Starting with an illegitimate state with an ISTS, subsequent system states will have an ISTS until the root node is active; after that action all subsequent system states will have a FSTS (Remark 1).
- If at any state the FSTS contains all the nodes of the graph, the state is a legitimate state, i.e., we have a spanning tree of the graph defined by the predecessor pointers.

Lemma 4 *Any node $i \neq r$ in ISTS for a system state will remain in the ISTS for all subsequent system states until the root node takes action.*

Proof : No node in an ISTS is privileged other than the root; in order to be privileged again, its predecessor must be active first. Thus, all members in an ISTS remain in ISTS until the root node r is active; when the node r in an ISTS takes action, the STS for the resulting system state may consist of the root node alone, if the level of the root node is changed by the action. □

Lemma 5 *Any node i in the FSTS for a system state will always remain in the FSTS for all subsequent system states.*

Proof : Similar to the proof of the previous lemma. □

Remark 5 *In general, the cardinality of the set STS is non decreasing upto a certain point, then drops to 1 (when the root node takes action) and then becomes non decreasing again with each subsequent system state.*

Definition 5 *Let T be an arbitrary set of nodes in G that does not contain the root node. We call an action taken by a node $i \in T$ a* **T-closed action** *iff either the action does not change $P(i)$ or the action makes $P(i) \in T$. For a given set T at any given system state, define $\ell_{min} = \min\{L(i) | i \in T\}$, $T_{min} = \{i | L(i) = \ell_{min}\}$ and $T' = T - T_{min}$.*

Lemma 6 *For any given set T, $r \notin T$, there cannot exist an infinite sequence of T-closed actions.*

Proof : We use induction on $|T|$. When $|T| = 1$, the node in T can take a T-closed action by making only a type I move; but this can be done at most n times after which no more T-closed action is possible. Hence, the claim is true for $|T| = 1$. Assume the claim is true for $|T| = k - 1$ for some $k,\ 1 < k < n$. Let $|T| = k$. Assume that an infinite sequence of T-closed actions exists. Note that any move by nodes in T' does not change the set T_{min}. First, we show that some node(s) in T_{min} must make a move in finite time. Any move by a node in T', if not T'-closed, has to be a type II move and the new predecessor will be in T_{min} (the move is T-closed by assumption) and hence the node remains un privileged (Remark 2) until nodes in T_{min} make a move. Since $|T'| < |T|$, by induction hypothesis, there is no infinite sequence of T'-closed moves. Hence some node in T_{min} will make a move in finite time and to be T-closed, nodes in T_{min} can make only type I moves. If all nodes in T_{min} make concurrent type I moves, ℓ_{min} will increase by 1 and we get a new T_{min} set, or else $|T_{min}|$ will decrease. Repeating the same

argument in the latter case, we see that ℓ_{min} will increase by 1 in finite time. Applying similar argument repeatedly, ℓ_{min} will eventually become n and $|T'|$ will be zero and no more T-closed moves will be possible. Hence the claim follows. □

Lemma 7 *In any illegitimate state with a FSTS (root node is not privileged), the FSTS grows in size after finitely many moves.*

Proof : Let $T = V - FSTS$ (V is the set of all nodes in the graph). If any node $i \in T$ takes an action which is not T-closed, then after this move the node i enters into $FSTS$. Since an infinite sequence of T-closed actions is impossible, FSTS grows in size in finitely many moves. □

Corollary 1 *In any illegitimate state with an ISTS (root node is privileged), either the ISTS grows in size or the root node takes action in finitely many moves.*

Combining the above two lemmas we see that if the root node is privileged in any illegitimate state, it will take action (and become permanently un privileged) in finitely many moves and then subsequently the $FSTS$ will continually grow in size and will eventually consist of all nodes in the graph in finitely many moves. Thus we state the following theorem.

Theorem 1 *Starting from any illegitimate state, the proposed algorithm brings back the system to a legitimate state in finite time.*

4 Conclusion

We have proposed a new self-stabilizing distributed algorithm that can maintain an arbitrary spanning tree in a connected graph. We have developed a new technique to prove the correctness of our algorithm that may prove useful in designing self-stabilizing algorithms for other graph theoretic problems.

References

[ADG92] A. Arora, S. Dolev, and M. Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1(1):11–18, 1992.

[BGW89] G. M. Brown, M. G. Gouda, and C. L. Wu. Token systems that self-stabilize. *IEEE Trans. Comput.*, 38(6):845–852, June 1989.

[Dij74] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.

[Dij86] E. W. Dijkstra. A belated proof of self-stabilization. *J. of Distributed Computing*, 1(1):5–6, 1986.

[FD92] M. Flatebo and A. K. Datta. Two-State self-Stabilizing algorithms. In *Proceedings of the IPPS-92*, California, June 1992.

[GH90] M. Gouda and T. Herman. Stabilizing unison. *Inf. Processing Letters*, 35(4):171–175, 1990.

[HC92] S.T. Huang and N.-S. Chen. A self-stabilizing algorithm for constructing breadth first trees. *Inf. Processing Letters*, 41:109–117, January 1992.

[Kes88] J. L. W. Kessels. An exercise in proving self-stabilization with a variant function. *Inf. Processing Letters*, 29(2):39–42, 1988.

[Sch93] M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1):45–67, March 1993.

[SS92] S. Sur and P. K. Srimani. A self-stabilizing distributed algorithm to construct BFS spanning tress of a symmetric graph. *Parallel Processing Letters*, 2(2,3):171–180, September 1992.