Information Processing Letters 114 (2014) 515–518

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

A 4*n*-move self-stabilizing algorithm for the minimal dominating set problem using an unfair distributed daemon ☆

Well Y. Chiu, Chiuyuan Chen *, Shih-Yu Tsai

Department of Applied Mathematics, National Chiao Tung University, Hsinchu 30010, Taiwan

ARTICLE INFO

Article history:
Received 29 April 2013
Received in revised form 19 March 2014
Accepted 10 April 2014
Available online 23 April 2014
Communicated by Tsan-sheng Hsu

Keywords:
Self-stabilizing algorithm
Fault tolerance
Distributed computing
Graph algorithm
Domination

ABSTRACT

A distributed system is self-stabilizing if, regardless of its initial state, the system is guaranteed to reach a legitimate (i.e., correct) state in finite time. In 2007, Turau proposed the first linear-time self-stabilizing algorithm for the minimal dominating set (MDS) problem under an unfair distributed daemon [9]; this algorithm stabilizes in at most 9*n* moves, where *n* is the number of nodes in the system. In 2008, Goddard et al. [4] proposed a 5*n*-move algorithm. In this paper, we present a 4*n*-move algorithm.

1. Introduction

Self-stabilization is a concept of designing a distributed system for transient fault toleration and was introduced by Dijkstra in [2]. A fundamental idea of self-stabilizing algorithms is that the distributed system may be started from an arbitrary state, and after finite time, the system will reach a *legitimate* (i.e., correct) state. Notice that the (*global*) state of a distributed system consists of the (*local*) state of every node (also called process) and the content of every communication channel (or communication register). A distributed algorithm is *self-stabilizing* if the following two properties hold: *convergence* and *closure*. The convergence property ensures that, starting from any illegitimate state, the distributed system reaches a legitimate state in finite time without any external intervention. The closure property ensures that, after convergence, the system remains in the set of legitimate states.

This paper follows the conventions used in [9]. In particular, every node executes the same program, and maintains and changes its own state based on its current state and the states of its neighbors. A node can change its state by making a move, that is, by changing the value of at least one of its local variables. Our algorithm requires locally unique identifiers (i.e., no closed neighborhood contains two identical identifiers) and assumes the *shared memory* model of communication, where neighboring nodes can communicate via common variables or registers. Notice that the shared memory models have two variants: state reading model and link-register model; we assume the former, in which each node can directly read the internal state of its neighbors.

The program of every node comprises a collection of rules of the form:

⟨precondition⟩ → ⟨statement⟩.

The precondition is a Boolean expression involving the states of the node and its neighbors. The statement updates the state of the node. The execution of a statement is

☆ This research was partially supported by the National Science Council of the Republic of China under the grant NSC100-2115-M-009-004-MY2.

* Corresponding author.
E-mail addresses: weeeeeeeeell@gmail.com (W.Y. Chiu), cychen@mail.nctu.edu.tw (C. Chen).

http://dx.doi.org/10.1016/j.ipl.2014.04.011
0020-0190/

called a *move*. A rule is *enabled* (or *privileged*) if its precondition evaluates to be true. A node is *enabled* (or *privileged*) if at least one of its rules is enabled. It is assumed that rules are atomically executed, that is, the evaluation of a precondition and the move are performed in one atomic step. More precisely, we use *composite atomicity*, meaning that a node may read all its input variables, perform a state transition, and write all its output variables in a single atomic step.

Various execution models have been used in self-stabilizing algorithms and these models are encapsulated within the notion of a daemon (or scheduler): the *central daemon*, the *synchronous daemon*, and the *distributed daemon*. A daemon can be *fair* or *unfair*. Refer to [6] for these definitions. It is well-known that an unfair distributed daemon is more practical for implementations than the other types of daemons, and it is the daemon used in this paper and [4,9].

In this paper, we consider a distributed system whose topology is represented by an undirected, simple graph $G = (V, E)$, whose V represents the set of nodes and E represents the set of edges (i.e., interconnections between nodes). Let $n = |V|$. If two nodes are connected by an edge, they are called *neighbors*. A subset $S \subseteq V$ is a *dominating set* of G if every node of G is either a member of S or adjacent to a member of S. A dominating set of G is a *minimal dominating set* (MDS) of G if none of its proper subsets is a dominating set of G. An MDS has an application of clustering in wireless networks and is maintained for minimizing the number of required resource centers [7]. The MDS problem is that of finding an MDS of a given graph and this is the problem concerned in this paper.

We now briefly review previous results. In [8], Hedetniemi et al. proposed the first self-stabilizing algorithm for the MDS problem; their algorithm assumes the central daemon. In [11], Xu et al. presented an algorithm under the synchronous daemon. In [9], Turau proposed a $9n$-move algorithm under an unfair distributed daemon; this algorithm is the first linear-time self-stabilizing algorithm for the MDS problem. In [4], Goddard et al. presented a $5n$-move algorithm. A good survey for self-stabilizing algorithms for the MDS problem can be found in [6].

The time complexity of a self-stabilizing algorithm is estimated in terms of moves or in terms of rounds. As was mentioned in [9], for a wireless system with bounded resources, the number of moves is at least as important as the number of rounds. The reason is that a node has to broadcast the state to its neighbors after making a move and therefore a reduction of the number of moves prolongs the lifetime of the network. The paper [11] uses the number of rounds to estimate the time complexity; but [4,8,9] and this paper uses the number of moves. The main contribution of this paper is to propose a $4n$-move self-stabilizing algorithm for the MDS problem under an unfair distributed daemon. We now summarize all the known results in Table 1.

This paper is organized as follow. In Section 2, we present a $4n$-move self-stabilizing algorithm for the MDS problem. Concluding remarks are given in Section 3. A preliminary version of this paper appeared as [1].

Table 1
Self-stabilizing algorithms for the MDS problem.

	stabilization time	daemon type
Hedetniemi et al. [8]	$(2n+1)n$ moves	central
Xu et al. [11]	$4n$ rounds	synchronous
Turau [9]	$9n$ moves	distributed
Goddard et al. [4]	$5n$ moves	distributed
this paper	$4n$ moves	distributed

2. Main result: a $4n$-move algorithm

The purpose of this section is to present our main result: $\mathcal{A}_{\text{MDS}}$, a $4n$-move self-stabilizing algorithm for the MDS problem under an unfair distributed daemon. $\mathcal{A}_{\text{MDS}}$ uses four (local) states, which are defined by the four-valued variable *state*. The range of values of *state* is: `IN`, `OUT1`, `OUT2`, and `WAIT`. A node with $state = \texttt{IN}$ will be referred to as an `IN` node. A neighbor is an `IN` neighbor if it is an `IN` node. Let $S = \{v : v.state = \texttt{IN}\}$. Nodes with $state = \texttt{OUT1}$ or `OUT2` or `WAIT` will be referred to as `OUT` nodes.

The values of *state* have the following meaning. The value `IN` indicates that the node is in the MDS. The value `OUT1` means that the node is not in the MDS and it has a unique `IN` neighbor. The value `OUT2` indicates that the node is not in the MDS and it has at least one `IN` neighbor. The value `WAIT` means that the node is not in the MDS and it does not have any `IN` neighbor. To make it precise, in our self-stabilizing MDS algorithm, a *legitimate state* is: the set of `IN` nodes form an MDS, every `OUT` node with $state = \texttt{OUT1}$ has a unique `IN` neighbor, every `OUT` node with $state = \texttt{OUT2}$ has at least one `IN` neighbor, and there is no node with $state = \texttt{WAIT}$.

Let $N(v)$ denote the set of neighbors of node v and $v.id$ denote the identifier of v. To formally define the rules of $\mathcal{A}_{\text{MDS}}$, the following predicates defined for each node v are needed:

- $noBtNbr \equiv \nexists u \in N(v) : u.state = \texttt{WAIT} \wedge u.id < v.id$.
- $noDpNbr \equiv \nexists u \in N(v) : u.state = \texttt{OUT1}$.

When two or more neighboring nodes want to enter the MDS simultaneously, our algorithm chooses the one with the smaller (smallest) *id*. According to this, if v is an `OUT` node and has a neighbor u such that $u.state = \texttt{WAIT}$ and $u.id < v.id$, then u is called a *better neighbor*. The predicate *noBtNbr* indicates that v has no better neighbor. Also, if v is an `IN` node and has a neighbor u with $u.state = \texttt{OUT1}$, then u is called a *dependent neighbor* since u depends on its unique neighbor in the MDS (the neighbor is v). The predicate *noDpNbr* indicates that v has no dependent neighbor.

For convenience, we introduce $InNbr = |\{u \in N(v) \mid u.state = \texttt{IN}\}|$. The algorithm $\mathcal{A}_{\text{MDS}}$ uses the following six rules and its state diagram is given in Fig. 1.

R1. $state = \texttt{WAIT} \wedge InNbr = 0 \wedge noBtNbr \rightarrow state := \texttt{IN}$.
R2. $state = \texttt{IN} \wedge InNbr = 1 \wedge noDpNbr \rightarrow state := \texttt{OUT1}$.
R3. $state = \texttt{IN} \wedge InNbr > 1 \wedge noDpNbr \rightarrow state := \texttt{OUT2}$.
R4. $state = \texttt{WAIT} \wedge InNbr = 1 \rightarrow state := \texttt{OUT1}$.

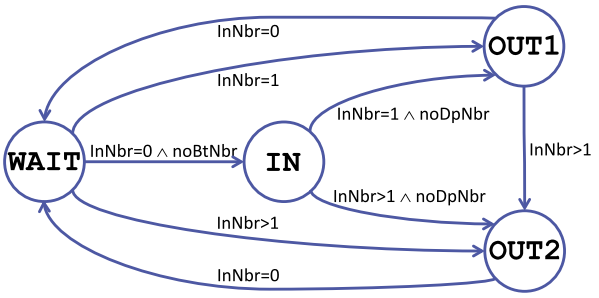

Fig. 1. The state diagram of $\mathcal{A}_{\text{MDS}}$.

R5. $(state = \texttt{OUT1} \vee state = \texttt{WAIT}) \wedge InNbr > 1 \rightarrow state := \texttt{OUT2}$.

R6. $(state = \texttt{OUT1} \vee state = \texttt{OUT2}) \wedge InNbr = 0 \rightarrow state := \texttt{WAIT}$.

We now prove the correctness of $\mathcal{A}_{\text{MDS}}$.

Lemma 2.1. *In any state in which no node is enabled the set S is a minimal dominating set of G.*

Proof. Suppose to the contrary that S is not a minimal dominating set of G. Then either (i) S is not a dominating set or (ii) S is a dominating set but not minimal. First consider (i). Since S is not a dominating set, there exists at least one node $u \notin S$ which has no IN neighbor; let S' be the set of all such nodes. Since rule 6 is not enabled, every node in S' has $state = \texttt{WAIT}$. Let u_0 be the node in S' with minimum id. Then u_0 satisfies all the constraints of rule 1. Hence, rule 1 is enabled and this contradicts to the assumption that no node is enabled. Now consider (ii). Since S is a dominating set but not minimal, there must exist at least one node $u \in S$ such that $S \backslash \{u\}$ is also a dominating set of G. Then $|N(u) \cap S| \geq 1$ and for all u' in $N(u) \backslash S$, we have $|N(u') \cap S| \geq 2$. Thus, every node u' in $N(u) \backslash S$ has $InNbr > 1$. Hence, every node u' in $N(u) \backslash S$ must have $u'.state = \texttt{OUT2}$; otherwise rule 5 is enabled on u'. Consequently, node u has $noDpNbr = true$ and either $InNbr = 1$ (if $|N(u) \cap S| = 1$) or $InNbr > 1$ (if $|N(u) \cap S| > 1$). Hence, either rule 2 or rule 3 is enabled on node u, which is a contradiction. □

We now show that $\mathcal{A}_{\text{MDS}}$ converges in finite time. In particular, we show that the number of moves of $\mathcal{A}_{\text{MDS}}$ is at most $4n - 2$. Let k be a nonnegative integer and $\langle r_1, r_2, \ldots, r_k \rangle$ be a sequence of rules (r_i's are not necessarily distinct). The sequence $\langle r_1, r_2, \ldots, r_k \rangle$ is called a *move sequence* if a node can execute r_1, then r_2, ..., then r_k. The following two lemmas show that in any possible move sequence of a specific node, rule 1 and rule 6 will appear at most once.

Lemma 2.2. *If a node executes rule 1, then it will not execute any other rule. Consequently, if a node enters the set S, then it will never leave S.*

Proof. Let v be a node which executes rule 1. Then $v.state$ is set to IN and v enters S. By the precondition of rule 1, v has no IN neighbor and no better neighbor; therefore no neighbor of v enters S at the same time. Thus, no node in $N(v)$ enters S and therefore $InNbr = 0$. After executing rule 1, $v.state$ is IN and the possible rule that v can execute is either rule 2 or rule 3. Rule 2 is impossible since it requires $InNbr = 1$; rule 3 is also impossible since it requires $InNbr > 1$. Therefore, v will not execute any other rule. The second statement of this lemma now follows. □

Lemma 2.3. *A node can execute rule 6 at most once, or equivalently, a node can set its state to* WAIT *at most once.*

Proof. Let v be a node which executes rule 6. By the precondition of rule 6, v has no IN neighbor. After executing rule 6, $v.state$ is set to WAIT and the possible rule that v can execute is rule 1 or rule 4 or rule 5. If v executes rule 1, then by Lemma 2.2, v will not execute any other rule and we have this lemma. If v executes rule 4, then $InNbr = 1$ must be *true* before rule 4 is enabled, meaning that v has a neighbor (say, u) which has executed rule 1; by Lemma 2.2, u will never leave S and therefore it is impossible to have $InNbr = 0$, which means that v cannot execute rule 6 again. Similarly, if v executes rule 5, then $InNbr > 1$ must be *true* before rule 5 is enabled, meaning that v has two neighbors (say, u and w) which have executed rule 1; by Lemma 2.2, both u and w will never leave S and therefore it is impossible to have $InNbr = 0$, which means that v cannot execute rule 6 again. □

Theorem 2.4. *The proposed algorithm $\mathcal{A}_{\text{MDS}}$ is self-stabilizing under an unfair distributed daemon and it stabilizes after at most $4n - 2$ moves with a minimal dominating set, where n is the number of nodes. Moreover, the bound $4n - 2$ is tight.*

Proof. By Lemma 2.1, $\mathcal{A}_{\text{MDS}}$ is correct. To prove that $\mathcal{A}_{\text{MDS}}$ stabilizes after at most $4n - 2$ moves, we first prove that it stabilizes after at most $4n$ moves. To do this, it suffices to show that any move sequence of a node is of length at most 4 under an unfair distributed daemon. Let v be an arbitrary node in G. By Lemma 2.3, v can execute rule 6 at most once. Thus, there are two cases: v never executes rule 6 and v executes rule 6 once.

First consider the case that v never executes rule 6. Then $v.state$ never changes to WAIT. Thus, the move sequence of v is either $\langle 1 \rangle$ or $\langle 2, 5 \rangle$ or $\langle 4, 5 \rangle$. It follows that any move sequence of v is of length at most 2.

Now consider the case that v executes rule 6 once. In this case, regard a move sequence of v as the concatenation of a prefix and a suffix. By Lemma 2.2, the prefix of any move sequence of v cannot contain 1 since if v executes rule 1 then v will not execute any other rule, including rule 6. Hence, the possible prefix of any move sequence of v is either $\langle 2, 6 \rangle$ or $\langle 3, 6 \rangle$ or $\langle 4, 6 \rangle$ or $\langle 5, 6 \rangle$. After v executes rule 6, $v.state$ changes to WAIT. Thus, the possible suffix of any move sequence of v is either $\langle 6, 1 \rangle$ or $\langle 6, 4, 5 \rangle$ or $\langle 6, 5 \rangle$. Concatenating the prefix and suffix, we conclude that any move sequence of v is of length at most 4.

From the above, $\mathcal{A}_{\text{MDS}}$ stabilizes after at most $4n$ moves with a minimal dominating set. We now prove that the

bound can be strengthened to $4n-2$. The cases of $n=1$ and $n=2$ are trivial. Suppose $n \geq 3$ and one of the nodes makes 4 moves; by the above argument, this node has two neighbors executing rule 1. Thus, at least two nodes in G make less than 4 moves and the upper bound can be strengthened to $4n-2$. We now prove that this bound is tight. Consider the complete bipartite graph $K_{2,n-2}$, where $n \geq 3$. Let the two nodes in the partite of cardinality two have the maximum and the minimum identifiers among the n nodes. If initially all nodes are in state IN, then there is a way that all the rest of the nodes executes $\langle 3, 6, 4, 5\rangle$ but nodes with the maximum and minimum identifiers execute $\langle 3, 6, 1\rangle$. All together $4n-2$ moves are made. □

Before ending this section, we would like to point out an error in the modified MDS algorithm in [9] (denote it as $\mathcal{A}'_{\text{MDS}}$ here). The author claimed (in page 93) that rule 4 can be changed by replacing the predicate *inNeighbor* with *inNeighborWithLowerId* so that the total number of moves can be further reduced. We now show that the author's claim is incorrect. Suppose the replacement is done and the resultant rule is called rule 4′. Let G be a path of three nodes $v_1 - v_2 - v_3$. Suppose $v_i.id$ is i and suppose the initial states are: v_1, v_2 are IN nodes with $dependent = \Lambda$ and v_3 is an OUT node with $dependent = v_2$. Since $\{v_1, v_2\}$ is not an MDS, algorithm $\mathcal{A}'_{\text{MDS}}$ must make a move. It is easy to verify that no rule can be enabled by $\mathcal{A}'_{\text{MDS}}$.

3. Concluding remarks

The main result of this paper is a $(4n-2)$-move self-stabilizing algorithm for the MDS problem using an unfair distributed daemon and the bound $4n-2$ is tight. The previous best algorithm is a $5n$-move algorithm. The model that we use is the normal model, also called the distance-1 model. Notice that in [3] and [10], the authors considered the distance-2 model, in which every node can read the states of nodes up to distance 2; see also [5] for the distance-k model.

Acknowledgements

The authors would like to greatly thank the referees for their constructive comments and suggestions that greatly improve the presentation of this paper.

References

[1] W.Y. Chiu, C. Chen, Linear-time self-stabilizing algorithms for minimal domination in graphs, in: Proceedings of the International Workshop on Combinatorial Algorithms (IWOCA), in: Lect. Notes Comput. Sci., vol. 8288, 2013, pp. 115–126.
[2] E.W. Dijkstra, Self-stabilizing systems in spite of distributed control, Commun. ACM 17 (11) (1974) 643–644.
[3] M. Gairing, W. Goddard, S.T. Hedetniemi, P. Kristiansen, A.A. McRae, Distance-two information in self-stabilizing algorithms, Parallel Process. Lett. 14 (3–4) (2004) 387–398.
[4] W. Goddard, S.T. Hedetniemi, D.P. Jacobs, P.K. Srimani, Z. Xu, Self-stabilizing graph protocols, Parallel Process. Lett. 18 (1) (2008) 189–199.
[5] W. Goddard, S.T. Hedetniemi, D.P. Jacobs, V. Trevisan, Distance-k knowledge in self-stabilizing algorithms, Theor. Comput. Sci. 399 (2008) 118–127.
[6] N. Guellati, H. Kheddouci, A survey on self-stabilizing algorithms for independent, domination, coloring, and matching in graphs, J. Parallel Distrib. Comput. 70 (4) (2010) 406–415.
[7] T.W. Haynes, S.T. Hedetniemi, P.J. Slater, Fundamentals of Domination in Graphs, Marcel Dekker, 1998.
[8] S.M. Hedetniemi, S.T. Hedetniemi, D.P. Jacobs, P.K. Srimani, Self-stabilizing algorithms for minimal dominating sets and maximal independent sets, Comput. Math. Appl. 46 (5–6) (2003) 805–811.
[9] V. Turau, Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler, Inf. Process. Lett. 103 (3) (2007) 88–93.
[10] V. Turau, Efficient transformation of distance-2 self-stabilizing algorithms, J. Parallel Distrib. Comput. 72 (2012) 603–612.
[11] Z. Xu, S.T. Hedetniemi, W. Goddard, P.K. Srimani, A synchronous self-stabilizing minimal domination protocol in an arbitrary network graph, in: Proc. 5th International Workshop on Distributed Computing, 2003.