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

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

1. Introduction

Self-stabilization is a concept of designing a distributed system for transient fault tolerance 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>.

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: weeeweell@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/© 2014 Elsevier B.V. All rights reserved.
called a move. A rule is enabled (or privileged) if its precondition evaluates to 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 4m-move self-stabilizing algorithm for the MDS problem under an unfair distributed daemon. We now summarize all the known results in Table 1.

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Self-stabilizing algorithms for the MDS problem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hedetniemi et al. [8]</td>
<td>(2n + 1)n moves</td>
</tr>
<tr>
<td>Xu et al. [11]</td>
<td>4n moves</td>
</tr>
<tr>
<td>Turau [9]</td>
<td>9n moves</td>
</tr>
<tr>
<td>Goddard et al. [4]</td>
<td>5n moves</td>
</tr>
<tr>
<td>this paper</td>
<td>4n moves</td>
</tr>
</tbody>
</table>

2. Main result: a 4m-move algorithm

The purpose of this section is to present our main result: \( A_{MDS} \), a 4m-move self-stabilizing algorithm for the MDS problem under an unfair distributed daemon. \( A_{MDS} \) 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 = \( IN \) will be referred to as an \( IN \) node. A neighbor is an \( IN \) neighbor if it is an \( IN \) node. Let \( S = \{ v : v.state = IN \} \). Nodes with state = \( OUT1 \) 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 = \( OUT1 \) has a unique \( IN \) neighbor, every \( OUT \) node with state = \( OUT2 \) has at least one \( IN \) neighbor, and there is no node with state = \( WAIT \).

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 \( A_{MDS} \), the following predicates defined for each node \( v \) are needed:

- \( noBtNbr \equiv \exists u \in N(v) : u.state = WAIT \land u.id < v.id. \)
- \( noDpNbr \equiv \exists u \in N(v) : u.state = OUT1. \)

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 \( u.state = WAIT \) 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 \( u.state = OUT1 \), 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 \( InNbr = \{ u \in N(v) \mid u.state = IN \} \). The algorithm \( A_{MDS} \) uses the following six rules and its state diagram is given in Fig. 1.

R1. \( state = WAIT \land InNbr = \emptyset \land noBtNbr \Rightarrow state := IN \).  
R2. \( state = IN \land InNbr = 1 \land noDpNbr \Rightarrow state := OUT1 \).  
R3. \( state = IN \land InNbr > 1 \land noDpNbr \Rightarrow state := OUT2 \).  
R4. \( state = WAIT \land InNbr = 1 \Rightarrow state := OUT1 \).  

This paper is organized as follow. In Section 2, we present a 4m-move self-stabilizing algorithm for the MDS problem. Concluding remarks are given in Section 3. A preliminary version of this paper appeared as [1].
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 \textit{IN} neighbor; let \( S' \) be the set of all such nodes. Since rule 6 is not enabled, every node in \( S' \) has state \( \text{WAIT} \). 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 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 \( S \setminus \{u\} \) is also a dominating set of \( G \). Then \( |N(u) \cap S| \geq 1 \) and for all \( u' \in N(u) \setminus S \), we have \( |N(u') \cap S| \geq 2 \). Thus, every node \( u' \in N(u) \setminus S \) has \( \text{INnbr} > 1 \). Hence, every node \( u' \in N(u) \setminus S \) must have \( u'.\text{state} = \text{OUT2} \); otherwise rule 5 is enabled on \( u' \). Consequently, node \( u \) has \( \text{noINbr} = \text{true} \) and either \( \text{INnbr} = 1 \) (if \( |N(u) \cap S| = 1 \)) or \( \text{INnbr} > 1 \) (if \( |N(u) \cap S| > 1 \)). Hence, either rule 2 or rule 3 is enabled on node \( u \), which is a contradiction. \( \square \)

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 \( \langle r_i \rangle \)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 \( r_2 \), \ldots, 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 \( v.\text{state} \) is set to \textit{IN} and \( v \) enters \( S \). By the precondition of rule 1, \( v \) has no \textit{IN} neighbor and no better neighbor; therefore no neighbor of \( v \) enters \( S \) at the same time. Thus, no node in \( \text{N}(v) \) enters \( S \) and therefore \( \text{INnbr} = 0 \). After executing rule 1, \( v.\text{state} \) is \textit{IN} and the possible rule that \( v \) can execute is either rule 2 or rule 3. Rule 2 is impossible since it requires \( \text{INnbr} = 1 \); rule 3 is also impossible since it requires \( \text{INnbr} > 1 \). Therefore, \( v \) will not execute any other rule. The second statement of this lemma now follows. \( \square \)

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

**Proof.** Let \( v \) be a node which executes rule 6. By the precondition of rule 6, \( v \) has no \textit{IN} neighbor. After executing rule 6, \( v.\text{state} \) is set to \textit{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 \( \text{INnbr} = 1 \) must be \textit{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 \( \text{INnbr} = 0 \), which means that \( v \) cannot execute rule 6 again. Similarly, if \( v \) executes rule 5, then \( \text{INnbr} > 1 \) must be \textit{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 \( \text{INnbr} = 0 \), which means that \( v \) cannot execute rule 6 again. \( \square \)

**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 \( v.\text{state} \) never changes to \textit{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, \( v.\text{state} \) changes to \textit{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 $\langle 3,6,1 \rangle$. Altogether $4n - 2$ 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 $A'_MDS$ here). The author claimed (in page 93) that rule 4 can be changed by replacing the predicate $\text{inNeighbor}$ with $\text{inNeighborWithLowerId}$ 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 $\text{dependent} = \Lambda$ and $v_3$ is an OUT node with $\text{dependent} = v_2$. Since $\{v_1, v_2\}$ is not an MDS, algorithm $A'_MDS$ must make a move. It is easy to verify that no rule can be enabled by $A'_MDS$.

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