A self-stabilizing algorithm for the bridge-finding problem assuming the distributed demon model

Tetz C. Huang
Department of Computer Science and Engineering, Yuan-Ze University, 135 Yuan-Tung Road, Chung-Li, Tao-Yuan 320, Taiwan
E-mail:cstetz@saturn.yzu.edu.tw

Ji-Cherng Lin
Department of Computer Science and Engineering, Yuan-Ze University, 135 Yuan-Tung Road, Chung-Li, Tao-Yuan 320, Taiwan
E-mail:csjclin@saturn.yzu.edu.tw

Cheng-Pin Wang
Department of Computer Science and Engineering, Yuan-Ze University, 135 Yuan-Tung Road, Chung-Li, Tao-Yuan 320, Taiwan
E-mail:s919403@mail.yzu.edu.tw

Chih-Yuan Chen
Department of Computer Science and Information Engineering, Nanya Institute of Technology, 414, Sec. 3, Chung-Shan East Road, Chung-Li, Tao-Yuan 320, Taiwan
E-mail:cychen@nanya.edu.tw

Abstract

A bridge is an edge whose deletion causes a distributed system to become disconnected. Thus bridges are edges critical to distributed system reliability. In this paper, we propose a self-stabilizing algorithm that can find all bridges in a distributed system. Under the assumption that a BFS tree have been constructed in the system, our algorithm is self-stabilizing not only under the central demon model but also under the more general distributed demon model. The worst-case stabilization time of our algorithm under the distributed demon model is at most n^2 steps, where n is the number of nodes in the system.

Keywords: Self-stabilizing algorithm, Bridge finding problem, Distributed demon model.

1 Introduction

A *distributed system* consists of a set of loosely connected processors that do not share a global memory. It is usually modelled by a connected simple undirected graph $G = (V, E)$, with each node $x \in V$ representing a processor in the system and each edge $\{x, y\} \in E$ representing the link connecting processors x and y. In the rest of this paper, the terms node and processor will be used interchangeably. In any distributed system considered in this paper, each processor has a set of *shared registers* (*registers* for short). For each register r_x in a processor x, only x can write values into it and only x and its neighbors can read values of it. Each processor is equipped with a *local algorithm* that consists of one or more rules of the form:

condition part $\rightarrow$ *action part*.

The condition part (or *guard*) is a Boolean expression of registers of the processor and its neighbors, and the action part is an assignment of values to some registers of the processor. If the condition part of one or more rules in a processor is evaluated as true, we say that the processor is *privileged* to execute the action part of any of these rules (or *privileged to make a move*). The local algorithms of all processors in a distributed system constitute a *distributed algorithm*. The *local state* of a processor is specified by the values of all its shared registers. The local states of all processors in the system constitute a *global configuration* (*configuration* for short) of the system. Normally, a distributed algorithm is designed to solve a specific problem such as the shortest path problem, the mutual exclusion problem, etc. According to the problem to be solved, *legitimate configurations* are defined in such a way that if the system is in a legitimate configuration, the solution of the problem can be seen.

1.1 The computational models

The *central demon model* was first introduced by Dijkstra [4] in 1974. Under this computational model, if the system starts with a configuration in which no node in

the system is privileged, then the system is deadlocked. Otherwise, the *central demon* in the system will randomly select exactly one privileged processor and exactly one rule in the processor, and let the selected processor execute the action part of the selected rule. The local state of the activated processor thus changes, which in the meantime results in the change of the configuration of the system. The system will then repeat the above process to change configurations as long as it does not encounter any deadlock situation. Thus the behavior of the system under the action of the algorithm can be described by *executions*. An infinite or finite sequence of configurations $\Gamma = (\gamma_1, \gamma_2, \ldots)$ of a distributed system is called an *execution* (of the algorithm in the system) *under the central demon model* if for any $i \geq 1$, γ_{i+1} is obtained from γ_i after exactly one processor in the system makes the i^{th} *move* $\gamma_i \rightarrow \gamma_{i+1}$, and in the case that Γ is finite, it is further required that no node is privileged in the last configuration. An algorithm is defined to be *self-stabilizing* if every execution of the algorithm has a suffix in which all configurations are legitimate. (Note that although this definition of self-stabilizing algorithms is not sufficient for some cases such as the self-stabilization with respect to the mutual exclusion problem, it does apply to most cases, including the self-stabilization with respect to the bridge-finding problem to be studied in this paper.)

The more general *distributed demon model* was later considered by Burns [1] in 1987. The difference between the central demon model and the distributed demon model is the number of processors that join in the execution of each atomic step of the system. Under the central demon model, exactly one privileged processor in the system is randomly selected by the central demon to execute its local algorithm in an atomic step. Under the distributed demon model, however, an arbitrary number of privileged processors are randomly selected by the distributed demon to execute their local algorithms simultaneously in an atomic step. In the case of distributed demon model, an atomic step of the system will also be called a *system move* in this paper. For each of the processors activated by the distributed demon to execute rules in a system move, we say that it *joins in the system move*. Under the distributed demon model, the behavior of the system under the action of the algorithm can also be described by *executions*. An infinite or finite sequence of configurations $\Gamma = (\gamma_1, \gamma_2, \ldots)$ of a distributed system is called an *execution* (of the algorithm in the system) *under the distributed demon model* if for any $i \geq 1$, γ_{i+1} is obtained from γ_i after a certain number of privileged processors selected by the distributed demon collectively make the i^{th} *system move* $\gamma_i \rightarrow \gamma_{i+1}$, and in the case that Γ is finite, it is further required that no node is privileged in the last configuration. The definition for an algorithm to be *self-stabilizing under the distributed demon model* is the same as under the central demon model. Since "selecting an arbitrary number of privileged processors" includes "selecting exactly one privileged processor" as a special case, if a system is self-stabilizing under the distributed demon model, then it is self-stabilizing under the central demon model. However, the converse, as is well-known, is not true.

1.2 Self-stabilizing bridge-finding algorithms

Three self-stabilizing bridge-finding algorithms have been proposed in the past: the SBF Algorithm in Karaata and Chaudhuri [5], the phase I part of Algorithm A in Chaudhuri [2], and Algorithm UNNS in Devismes [3].

In [5], under the assumption that a BFS tree has been constructed in a distributed system, Theorem 13 claims that the SBF Algorithm is a self-stabilizing bridge-finding algorithm under the weakly fair demon model, and Theorem 14 further claims that the SBF Algorithm is self-stabilizing under the (non-fair) central demon model. After a BFS tree has been constructed in the system, the SBF algorithm in [5] requires $O(n^2 |E|)$ steps to stabilize under the central demon model, where n and $|E|$ are the number of nodes and the number of edges in the system, respectively.

Under the assumption that a DFS tree has been constructed in a distributed system, Lemma 4.17 in [2] and Lemma 2 in [3] show, respectively, that the phase I part of Algorithm A and Algorithm UNNS are self-stabilizing bridge-finding algorithms under the central demon model. After a DFS tree has been constructed, both of these two self-stabilizing bridge-finding algorithms require $O(n^2)$ to stabilize under the central demon model.

As mentioned above, all the existing self-stabilizing bridge-finding algorithms work under the central demon model. However, to the best of our knowledge, no self-stabilizing bridge-finding algorithm under the distributed demon model have been proposed in the past.

1.3 Our contributions

In this paper, we propose a self-stabilizing algorithm that can find all bridges in a distributed system. Under the assumption that a BFS tree has been constructed in the system, our algorithm is self-stabilizing not only under the central demon model but also under the more general distributed demon model. Assuming

that a BFS tree has been constructed, we have also computed the worst-case stabilization time of the algorithm under the distributed demon model, which is at most n^2 steps, where n is the number of nodes in the system.

1.4 Organization of this paper

The rest of this paper is organized as follows: In Section 2, our algorithm is presented, and the meaning of legitimate configurations is clarified - explicitly, it is shown that in any legitimate configuration, all bridges can be identified. In Section 3, a proof is given to show that the proposed algorithm is self-stabilizing under the distributed demon model. Moreover, assuming that a BFS tree has been constructed, the worst-case stabilization time under the distributed demon model is computed. Section 4 are some concluding remarks concerning a self-stabilizing bridge-finding algorithm that works without assuming the presence of any rooted tree.

2 The proposed algorithm and the legitimate configuration

In this section, our self-stabilizing algorithm for solving the bridge-finding problem will be presented. The distributed system in consideration has a general underlying topology, and can be modelled by a connected simple undirected graph $G = (V, E)$, with each node $x \in V$ representing a processor in the system and each edge $\{x, y\} \in E$ representing the bidirectional link connecting processors x and y. It is assumed that

(1) each node in the system has a unique identity and there is a special node r in the system,
(2) a BFS spanning tree (namely, BFS_G) of G, rooted at r, has been constructed,
(3) each node x in the system maintains a shared register b_x,
(4) $N(x)$ denotes the set of all neighbors of x, and
(5) the value of b_x is taken from the power set of $E \cup \overline{E}$, where $\overline{E} = \{\overline{\alpha} \mid \alpha \in E\}$ with $\overline{\alpha}$ being the simplified expression for the singleton $\{\alpha\}$.

For any node x, we define

$E(x) = \{\{x, y\} \mid y \in N(x)\}$, the set of all edges incident to x,
$C(x) = \{y \in N(x) \mid p_y = x\}$, the set of all *children of* x in BFS_G, where p_x stands for the parent of x in BFS_G,
$NT(x) = \{\{x, y\} \in E \mid y \notin C(x) \wedge x \notin C(y)\}$, the set of all *non-tree edges* in G (i.e., edges not in BFS_G) that are incident to x,
$One(x) = \{\alpha \in E \mid$ there is exactly one node $u \in C(x)$ s.t. $\alpha \in b_u\}$,
$Two(x) = \{\alpha \in E \mid$ there are exactly two nodes $u, v \in C(x)$ s.t. $\alpha \in b_u \wedge \alpha \in b_v\}$.

Algorithm 1
{For any node x}
$R1 : (B_1(x) \neq \emptyset \vee B_2(x) \neq \emptyset \vee B_3(x) \neq \emptyset \vee B_4(x) \neq \emptyset)$
$\rightarrow b_x := [b_x - (\overline{B_1}(x) \cup B_2(x) \cup B_3(x) \cup \overline{B_4}(x))] \cup B_1(x) \cup \overline{B_2}(x)$.

Note that in the above algorithm,

$B_1(x) = \{\alpha \in E \mid ([\alpha \in NT(x)] \vee [\alpha \in One(x) \wedge \alpha \notin E(x)]) \wedge \alpha \notin b_x\}$
$B_2(x) = \{\alpha \in E \mid \alpha \in Two(x) \wedge \alpha \notin E(x) \wedge \overline{\alpha} \notin b_x\}$
$B_3(x) = \{\alpha \in E \mid \alpha \in b_x \wedge \alpha \notin NT(x) \wedge \alpha \notin One(x)\}$
$B_4(x) = \{\alpha \in E \mid \overline{\alpha} \in b_x \wedge \alpha \notin Two(x)\}$
$\overline{B_i}(x) = \{\overline{\alpha} \mid \alpha \in B_i(x)\}$, for $i = 1$, 2 and 4.

Legitimate configurations of Algorithm 1 are defined to be all those configurations in each of which no node in the system is privileged. Thus, when the system is in a legitimate configuration γ, for any node x in V, $B_1(x) = B_2(x) = B_3(x) = B_4(x) = \emptyset$. The concepts of *ancestor*, *descendant*, *common ancestor* and *the lowest common ancestor* in BFS_G are as usually defined. Recall that for any two nodes x and y, the lowest common ancestor of x and y exists and is unique, which we shall denote by $lca(x, y)$. For any non-tree edge $\alpha = \{u, v\}$ in G, let B_u be the unique simple path in BFS_G connecting u and $lca(u, v)$ and B_v be the unique simple path in BFS_G connecting v and $lca(u, v)$. Note that the simple paths B_u and B_v and the edge $\{u, v\}$ forms a cycle in G. Let $\tau_{u,v}$ denote this cycle.

In the following, some graph-theoretic concepts and properties needed in later discussion are listed. Let $G = (V, E)$ be a simple connected graph. The *removal of an edge* $\{x, y\}$ from G results in the subgraph $G - \{x, y\}$ that consists of all nodes of G and all edges of G except $\{x, y\}$. An edge $\{x, y\}$ in G is called a *bridge in* G if $G - \{x, y\}$ is disconnected. The following properties are straightforward and hence the proofs are omitted.

Property 1 *An edge α is a bridge in G if and only if α does not lie on any cycle in G.*

Property 2 *Suppose $G' = (V, E')$ is a BFS spanning tree rooted at r of a simple connected graph $G = (V, E)$. If $\{x, y\} \in E - E'$ (i.e., $\{x, y\}$ is a non-tree edge), then x is not a descendant of y and y is not a descendant of x in G'.*

Property 3 *Suppose $G' = (V, E')$ is a rooted spanning tree for a simple connected graph $G = (V, E)$ and $\{x, y\}$ is an edge in E' (i.e., $\{x, y\}$ is a tree edge) with y being the child of x. If $\{x, y\}$ is not a bridge in G, then there exists an edge $\{u, v\}$ in $E - E'$ (i.e., $\{u, v\}$ is a non-tree edge) such that u is not a descendant of y and v is a descendant of y in G'.*

Lemma 1 *Suppose the system is in a legitimate configuration γ. For any node x, if $\alpha \in b_x$ or $\overline{\alpha} \in b_x$ in γ, then α is a non-tree edge in G, i.e., $\alpha \notin E_\gamma$.*

Proof. Suppose α is a tree edge in G.
Case 1. $\alpha \in b_x$ in γ. Since $B_3(x) = \emptyset$ in γ, $\alpha \notin B_3(x)$ in γ and hence $\alpha \in One(x)$ in γ. Hence there exists a child x_1 of x in BFS_G such that $\alpha \in b_{x_1}$ in γ. By the same token, there exists a child x_2 of x_1 in BFS_G such that $\alpha \in b_{x_2}$ in γ. Arguing in this way, we eventually get infinitely many nodes x, x_1, x_2, ... in the system such that x_{i+1} is a child of x_i in BFS_G for any $i = 1, 2, \ldots$. However, this is absurd because BFS_G is a rooted tree with only a finite number of nodes.
Case 2. $\overline{\alpha} \in b_x$ in γ. Since $B_4(x) = \emptyset$ in γ, $\alpha \notin B_4(x)$ in γ and hence $\alpha \in Two(x)$ in γ. Hence there exist two children u and v of x in BFS_G such that $\alpha \in b_u$ and $\alpha \in b_v$ in γ. By the same argument as in Case 1 above, we will get a contradiction from the condition that $\alpha \in b_u$.

Therefore, α must be a non-tree edge in G. ■

Lemma 2 *Suppose the system is in a legitimate configuration γ. Let $\alpha = \{u, v\}$ be a non-tree edge in G. If x is not an ancestor of either u or v in BFS_G, then $\alpha \notin b_x$ in γ.*

Proof. Suppose $\alpha \in b_x$ in γ. Since x is not an ancestor of either u or v in BFS_G, $x \neq u$ and $x \neq v$. Thus, $\alpha \notin NT(x)$ in γ. Since $B_3(x) = \emptyset$ in γ, we have that $\alpha \notin B_3(x)$. Since $[\alpha \notin NT(x) \wedge \alpha \in b_x \wedge \alpha \notin B_3(x)$ in $\gamma]$, we have that $\alpha \in One(x)$ in γ. Hence there exists a child x_1 of x such that $\alpha \in b_{x_1}$ in γ. Since x is not an ancestor of either u or v in BFS_G and x_1 is a descendant of x, we have that x_1 is not an ancestor of either u or v in BFS_G. By the same token, there exists a child x_2 of x_1 in BFS_G such that $\alpha \in b_{x_2}$ in γ, and x_2 is not an ancestor of either u or v in BFS_G. Arguing in this way, we will eventually get infinitely many nodes x_1, x_2, ... in the system such that x_{i+1} is a child of x_i in BFS_G for any $i = 1, 2, \ldots$. However, this is absurd because BFS_G is a rooted tree with only a finite number of nodes. Therefore, $\alpha \notin b_x$ in γ. ■

Lemma 3 *Suppose the system is in a legitimate configuration γ. For any node $x \in V$ and any edge $\alpha \in E$, b_x can not contain both α and $\overline{\alpha}$ in γ.*

Proof. Suppose both $\alpha \in b_x$ and $\overline{\alpha} \in b_x$ in γ. Since $B_4(x) = \emptyset$ in γ, we have that $\alpha \notin B_4(x)$ in γ. Since $[\alpha \notin B_4(x) \wedge \overline{\alpha} \in b_x$ in $\gamma]$, we have that $\alpha \in Two(x)$ in γ.
Case 1. $\alpha \notin NT(x)$ in γ. Since $\alpha \in Two(x)$ in γ, we have that $\alpha \notin One(x)$ in γ. Hence we have that $\alpha \in b_x \wedge \alpha \notin NT(x) \wedge \alpha \notin One(x)$ in γ, i.e., $\alpha \in B_3(x)$ in γ. Consequently, $B_3(x) \neq \emptyset$ in γ, which causes a contradiction.
Case 2. $\alpha \in NT(x)$ in γ. Let the non-tree edge α be $\{x, y\}$. Since $\alpha \in Two(x)$ in γ, there exists a child v of x in BFS_G such that $\alpha \in b_v$ in γ. By Lemma 2 and $\alpha \in b_v$ in γ, we have that either x or y is a descenant of v in BFS_G. Since v is a child of x in BFS_G, x is not a descendant of v in BFS_G. Thus, y is a descendant of v in BFS_G, and hence y is a descendant of x in BFS_G. Since $\{x, y\}$ is a non-tree edge in G and since BFS_G is a BFS tree rooted at r for G, this contradicts Property 2.

Therefore, b_x can not contain both α and $\overline{\alpha}$ in γ. ■

Lemma 4 *Suppose the system is in a legitimate configuration γ. Suppose $\alpha = \{u, v\}$ is a non-tree edge in G and the cycle $\tau_{u,v}$ in G is as previously defined prior to Lemma 1. If node x lies on $\tau_{u,v}$ and $x \neq lca(u, v)$, then $\alpha \in b_x$ in γ.*

Proof. Recall that $\tau_{u,v}$ is the cycle formed by B_u, B_v and $\alpha = \{u, v\}$, where B_u is the unique simple path in BFS_G connecting u and $lca(u, v)$, and B_v is the unique simple path in BFS_G connecting v and $lca(u, v)$. Let $B_u = (x_0, x_1, \ldots, x_t)$ with $x_0 = u$, $x_t = lca(u, v)$ and $t \geq 1$. Since $\alpha \in NT(x_0)$ in γ and $B_1(x_0) = \emptyset$ in γ, we have that $\alpha \in b_{x_0}$ in γ. If $t > 1$, then $x_1 \neq lca(u, v)$. For any child y of x_1 in BFS_G such that $y \neq x_0$, y cannot be an ancestor of either u or v in BFS_G. By Lemma 2, $\alpha \notin b_y$ in γ. This together with the fact that $\alpha \in b_{x_0}$ in γ implies that $\alpha \in One(x_1)$ in γ. Since it is obvious that $\alpha \notin E(x_1)$ and $B_1(x_1) = \emptyset$ in γ, we have that $\alpha \in b_{x_1}$ in γ. By the same token, if $t > 2$, then $\alpha \in b_{x_2}$ in γ. Arguing in this way, we will get that $\alpha \in b_{x_i}$ in γ for $i = 0, \ldots, t-1$. In other words, we have shown that for any node x in B_u except $lca(u, v)$, $\alpha \in b_x$ in γ. Similarly, we can show that for any node x in B_v except $lca(u, v)$, $\alpha \in b_x$ in γ. Hence the lemma is proved. ■

Lemma 5 *Suppose the system is in a legitimate configuration γ. Suppose $\alpha = \{u, v\}$ is a non-tree edge in G. For any node x in G, $\overline{\alpha} \in b_x$ in γ if and only if $x = lca(u, v)$ in BFS_G.*

Proof. ($\Leftarrow$) Suppose $x = lca(u, v)$ in BFS_G. For any child k of x in BFS_G which does not lie on $\tau_{u,v}$, k is not an ancestor of either u or v in BFS_G. Hence, by

Lemma 2, $\alpha \notin b_k$ in γ. Since $x = lca(u,v)$ in BFS_G, there are exactly two children u' and v' of x in BFS_G which lie on $\tau_{u,v}$. By Lemma 4, $\alpha \in b_{u'}$ and $\alpha \in b_{v'}$ in γ. Hence, $\alpha \in Two(x)$ in γ. Since $x = lca(u,v)$ in BFS_G and $\{u,v\}$ is a non-tree edge in G, $x \neq u$ and $x \neq v$ (for otherwise, u would be a descendant of v or v would be a descendant of u in BFS_G, which contradicts Property 2). Hence $\alpha \notin E(x)$ in G. This together with the fact that $\alpha \in Two(x)$ in γ and $B_2(x) = \emptyset$ in γ implies that $\overline{\alpha} \in b_x$ in γ.
($\Rightarrow$) Suppose $\overline{\alpha} \in b_x$ in γ. Since $B_4(x) = \emptyset$ in γ, $\alpha \in Two(x)$ in γ.

First, we show that x is a common ancestor of u and v in BFS_G. Since $\alpha \in Two(x)$ in γ, there exist two children y_1 and y_2 of x in BFS_G such that $\alpha \in b_{y_1}$ and $\alpha \in b_{y_2}$ in γ. By Lemma 2, y_1 is an ancestor of either u or v in BFS_G and so is y_2. Without loss of generality, we assume that y_1 is an ancestor of u in BFS_G. Thus, y_2 is not an ancestor of u in BFS_G (for otherwise, both y_1 and y_2 are ancestors of u in BFS_G and hence y_1 is a descendant of y_2 or y_2 is a descendant of y_1 in BFS_G, which contradicts the fact that y_1 and y_2 are two distinct children of x in BFS_G.). Hence y_2 must be an ancestor of v in BFS_G. Consequently, x is a common ancestor of u and v in BFS_G.

Next, we show that $x = lca(u,v)$ in BFS_G. Suppose $x \neq lca(u,v)$ in BFS_G. Since x is a common ancestor of u and v in BFS_G, x lies on the unique simple path P in BFS_G connecting $lca(u,v)$ and the root r. Since $x \neq lca(u,v)$ in BFS_G, there is exactly one child x_1 of x in BFS_G such that x_1 lies on P. This combined with the fact that $\alpha \in Two(x)$ in γ implies that there exists a child y of x in BFS_G such that y does not lie on P and $\alpha \in b_y$ in γ. Hence y is not an ancestor of either u or v in BFS_G and $\alpha \in b_y$ in γ. This contradicts Lemma 2. Therefore, $x = lca(u,v)$ in BFS_G. ■

Lemma 6 *Suppose the system is in a legitimate configuration γ. Let $\alpha = \{u,v\}$ be a non-tree edge in G. If node x lies on the unique simple path P in BFS_G connecting $lca(u,v)$ in BFS_G and the root r, then $\alpha \notin b_x$ in γ.*

Proof. Let $P = (x_0, \ldots, x_t)$ with $x_0 = lca(u,v)$, $x_t = r$ and $t \geq 0$. By Lemmas 3 and 5, we have that $\alpha \notin b_{x_0}$ in γ. If $t > 0$, then $x_1 \neq lca(u,v)$. Thus for any child y of x_1 in BFS_G such that $y \neq x_0$, y cannot be an ancestor of either u or v in BFS_G. By Lemma 4, $\alpha \notin b_y$ in γ. Hence $\alpha \notin One(x_1)$ in γ. This together with the fact that $\alpha \notin E(x_1)$ and $B_3(x_1) = \emptyset$ in γ implies that $\alpha \notin b_{x_1}$ in γ. By the same token, if $t > 1$ then $\alpha \notin b_{x_2}$ in γ. Arguing in this way, we will get that $\alpha \notin b_{x_i}$ in γ for $i = 1, \ldots, t$. Hence the lemma is proved. ■

Corollary 1 *Suppose the system is in a legitimate configuration γ. Let $\alpha = \{u,v\}$ be a non-tree edge in G. For any node x, $\alpha \in b_x$ in γ if and only if x lies on the cycle $\tau_{u,v}$ in G and $x \neq lca(u,v)$ in BFS_G.*

Proof. The "if" part is exactly Lemma 4, while the "only if" part follows immediately from Lemmas 2 and 6. ■

For presentation's sake, we use b_x^* to stand for the set $\{\alpha \in E \mid \alpha \in b_x \text{ or } \overline{\alpha} \in b_x\}$.

Corollary 2 *Suppose the system is in a legitimate configuration γ. Let $\alpha = \{u,v\}$ be a non-tree edge of G inγ. For any node $x \in V$, $\alpha \in b_x^*$ in γ if and only if x lies on the cycle $\tau_{u,v}$ in G.*

Proof. This follows immediately from Lemma 5 and Corollary 1. ■

The following theorem claims that in any legitimate configuration, all the bridges in G can be identified.

Theorem 1 *Suppose the system is in a legitimate configuration γ. A tree edge $\alpha = \{x,y\}$ (i.e., $\alpha \in E_\gamma$) is a bridge in G if and only if $b_x^* \cap b_y^* = \emptyset$ in γ.*

Proof. ($\Rightarrow$) Suppose $b_x^* \cap b_y^* \neq \emptyset$ in γ. Then there exists an edge $\beta = \{u,v\}$ in G such that $\beta \in b_x^* \cap b_y^*$ in γ. By Lemma 1, β is a non-tree edge in G. By Corollary 2, both x and y lie on the cycle $\tau_{u,v}$ in G. Hence, x and y can be connected by a simple path in BFS_G which lies on $\tau_{u,v}$. This simple path must be exactly the edge $\{x,y\}$ because $\{x,y\}$ is the unique simple path in BFS_G connecting x and y. Hence the edge $\{x,y\}$ must lie on the cycle $\tau_{u,v}$. By Property 1, $\{x,y\}$ cannot be a bridge in G.
($\Leftarrow$) Suppose $\alpha = \{x,y\} \in E_\gamma$ is not a bridge of G. Without loss of generality, we assume that x is the parent of y in BFS_G. By Property 3, there exists a non-tree edge $\beta = \{u,v\}$ such that u is not a descendant of y and v is a descendant of y in BFS_G. Let $P_{v,y}$ be the unique simple path in BFS_G connecting v and y. For any node z lying on the path $P_{v,y}$, $z \neq lca(u,v)$ (for if $z = lca(u,v)$, then u is a descendant of y, which causes a contradiction). Hence we have that $lca(u,v)$ is an ancestor of x in BFS_G. By Lemma 5 and Corollary 1, β or $\overline{\beta} \in b_x$ in γ, and $\beta \in b_y$ in γ. Hence $b_x^* \cap b_y^* \neq \emptyset$ in γ. ■

3 Correctness proof

In this section, we present a correctness proof, showing that Algorithm 1 is self-stabilizing under the distributed demon model. For convenience in presentation in the rest of this section, "execution" will mean "execution under the distributed demon model". The

following lemma is obvious in view of the definition of a legitimate configuration, the definition of a finite execution, and the definition of an algorithm being self-stabilizing under the distributed demon model.

Lemma 7 *Algorithm 1 is self-stabilizing under the distributed demon model if and only if any execution of Algorithm 1 is a finite execution.*

Lemma 8 *For any execution* Γ *of Algorithm* 1 *and any node* x *in the system, if* x *joins in a system move* $\gamma \rightarrow \gamma'$ *in* Γ *and all children of* x *in* BFS_G *does not join in* $\gamma \rightarrow \gamma'$*, then* x *is not privileged in* γ'*.*

Proof. Since all children of x in BFS_G does not join in $\gamma \rightarrow \gamma'$, the b-set of all children of x in BFS_G does not change in $\gamma \rightarrow \gamma'$ and hence the sets $One(x)$ and $Two(x)$ do not change due to $\gamma \rightarrow \gamma'$.

Claim 1. $B_1(x) = \emptyset$ in γ'.

Proof of claim. Let α be any edge.

Case 1. α does not satisfy $[\alpha \in NT(x) \vee (\alpha \in One(x) \wedge \alpha \notin E(x))$ in $\gamma']$. Then $\alpha \notin B_1(x)$ in γ'.

Case 2. α satisfies $[\alpha \in NT(x) \vee (\alpha \in One(x) \wedge \alpha \notin E(x))$ in $\gamma']$. Since the sets $NT(x)$, $E(x)$ and $One(x)$ do not change due to $\gamma \rightarrow \gamma'$, $\alpha \in NT(x) \vee (\alpha \in One(x) \wedge \alpha \notin E(x))$ in γ. Thus, $\alpha \notin B_2(x) \cup B_3(x)$ in γ.

Subcase 2.1. $\alpha \in b_x$ in γ. Then, since $\alpha \in b_x \wedge \alpha \notin B_2(x) \cup B_3(x)$ in γ, $\alpha \in b_x - (B_2(x) \cup B_3(x))$ in γ. Hence $\alpha \in [b_x - (\overline{B_1}(x) \cup B_2(x) \cup B_3(x) \cup \overline{B_4}(x))] \cup B_1(x) \cup \overline{B_2}(x)$ in γ. Since x executes $R1$ in $\gamma \rightarrow \gamma'$, $\alpha \in b_x$ in γ'. Hence $\alpha \notin B_1(x)$ in γ'.

Subcase 2.2. $\alpha \notin b_x$ in γ. Then, since $\alpha \in NT(x) \vee (\alpha \in One(x) \wedge \alpha \notin E(x))$ in γ and $\alpha \notin b_x$ in γ, $\alpha \in B_1(x)$ in γ. Hence $\alpha \in [b_x - (\overline{B_1}(x) \cup B_2(x) \cup B_3(x) \cup \overline{B_4}(x))] \cup B_1(x) \cup \overline{B_2}(x)$ in γ. Since x executes $R1$ in $\gamma \rightarrow \gamma'$, $\alpha \in b_x$ in γ'. Hence $\alpha \notin B_1(x)$ in γ'.

From all the above, we can conclude that $B_1(x) = \emptyset$ in γ'. ■

Claim 2. $B_2(x) = \emptyset$ in γ'.

Proof of claim. Let α be any edge.

Case 1. α does not satisfy $[\alpha \in Two(x) \wedge \alpha \notin E(x)$ in $\gamma']$. Then $\alpha \notin B_2(x)$ in γ'.

Case 2. α satisfies $[\alpha \in Two(x) \wedge \alpha \notin E(x)$ in $\gamma']$. Since the sets $Two(x)$ and $E(x)$ do not change due to in $\gamma \rightarrow \gamma'$, $\alpha \in Two(x) \wedge \alpha \notin E(x)$ in γ. Thus, $\alpha \notin B_1(x) \cup B_4(x)$ in γ and hence $\overline{\alpha} \notin \overline{B_1}(x) \cup \overline{B_4}(x)$ in γ.

Subcase 2.1. $\overline{\alpha} \in b_x$ in γ. Then, since $\overline{\alpha} \in b_x \wedge \overline{\alpha} \notin \overline{B_1}(x) \cup \overline{B_4}(x)$ in γ, $\overline{\alpha} \in [b_x - (\overline{B_1}(x) \cup B_2(x) \cup B_3(x) \cup \overline{B_4}(x))] \cup B_1(x) \cup \overline{B_2}(x)$ in γ. Since x executes $R1$ in $\gamma \rightarrow \gamma'$, $\overline{\alpha} \in b_x$ in γ'. Hence $\alpha \notin B_2(x)$ in γ'.

Subcase 2.2. $\overline{\alpha} \notin b_x$ in γ. Then, since $\alpha \in Two(x) \wedge \alpha \notin E(x) \wedge \overline{\alpha} \notin b_x$ in γ, $\alpha \in B_2(x)$ in γ and hence $\overline{\alpha} \in \overline{B_2}(x)$ in γ. Thus, $\overline{\alpha} \in [b_x - (\overline{B_1}(x) \cup B_2(x) \cup B_3(x) \cup \overline{B_4}(x))] \cup B_1(x) \cup \overline{B_2}(x)$ in γ. Since x executes $R1$ in $\gamma \rightarrow \gamma'$, $\overline{\alpha} \in b_x$ in γ'. Hence $\alpha \notin B_2(x)$ in γ'.

From all the above, we can conclude that $B_2(x) = \emptyset$ in γ'. ■

Claim 3. $B_3(x) = \emptyset$ in γ'.

Proof of claim. Let α be any edge.

Case 1. α does not satisfy $[\alpha \notin NT(x) \wedge \alpha \notin One(x)$ in $\gamma']$. Then $\alpha \notin B_3(x)$ in γ'.

Case 2. α satisfies $[\alpha \notin NT(x) \wedge \alpha \notin One(x)$ in $\gamma']$. Since the sets $NT(x)$ and $One(x)$ do not change due to $\gamma \rightarrow \gamma'$, $\alpha \notin NT(x) \wedge \alpha \notin One(x)$ in γ. Thus, $\alpha \notin B_1(x)$ in γ.

Subcase 2.1. $\alpha \in b_x$ in γ. Then, since $\alpha \notin NT(x) \wedge \alpha \notin One(x) \wedge \alpha \in b_x$ in γ, $\alpha \in B_3(x)$ in γ. Hence $\alpha \notin (b_x - B_3(x)) \cup B_1(x)$ in γ and hence $\alpha \notin [b_x - (\overline{B_1}(x) \cup B_2(x) \cup B_3(x) \cup \overline{B_4}(x))] \cup B_1(x) \cup \overline{B_2}(x)$ in γ. Since x executes $R1$ in $\gamma \rightarrow \gamma'$, $\alpha \notin b_x$ in γ'. Hence $\alpha \notin B_3(x)$ in γ'.

Subcase 2.2. $\alpha \notin b_x$ in γ. Then, since $\alpha \notin b_x \wedge \alpha \notin B_1(x)$ in γ, $\alpha \notin (b_x - B_3(x)) \cup B_1(x)$ in γ. Hence $\alpha \notin [b_x - (\overline{B_1}(x) \cup B_2(x) \cup B_3(x) \cup \overline{B_4}(x))] \cup B_1(x) \cup \overline{B_2}(x)$ in γ. Since x executes $R1$ in $\gamma \rightarrow \gamma'$, $\alpha \notin b_x$ in γ'. Hence $\alpha \notin B_3(x)$ in γ'.

From all the above, we can conclude that $B_3(x) = \emptyset$ in γ'. ■

Claim 4. $B_4(x) = \emptyset$ in γ'.

Proof of claim. Let α be any edge.

Case 1. α does not satisfy that $\alpha \notin Two(x)$ in γ'. Then $\alpha \notin B_4(x)$ in γ'.

Case 2. α satisfies that $\alpha \notin Two(x)$ in γ'. Since the set $Two(x)$ does not change due to $\gamma \rightarrow \gamma'$, $\alpha \notin Two(x)$ in γ. Thus, $\alpha \notin B_2(x)$ in γ.

Subcase 2.1. $\overline{\alpha} \in b_x$ in γ. Then, since $\alpha \notin Two(x) \wedge \overline{\alpha} \in b_x$ in γ, $\alpha \in B_4(x)$ in γ and hence $\overline{\alpha} \in \overline{B_4}(x)$ in γ. Hence $\overline{\alpha} \notin (b_x - \overline{B_4}(x)) \cup \overline{B_2}(x)$ in γ and hence $\overline{\alpha} \notin [b_x - (\overline{B_1}(x) \cup B_2(x) \cup B_3(x) \cup \overline{B_4}(x))] \cup B_1(x) \cup \overline{B_2}(x)$ in γ. Since x executes $R1$ in $\gamma \rightarrow \gamma'$, $\overline{\alpha} \notin b_x$ in γ'. Hence $\alpha \notin B_4(x)$ in γ'.

Subcase 2.2. $\overline{\alpha} \notin b_x$ in γ. Then, since $\overline{\alpha} \notin b_x \wedge \alpha \notin B_2(x)$ in γ, $\overline{\alpha} \notin (b_x - \overline{B_4}(x)) \cup \overline{B_2}(x)$ in γ. Hence $\overline{\alpha} \notin [b_x - (\overline{B_1}(x) \cup B_2(x) \cup B_3(x) \cup \overline{B_4}(x))] \cup B_1(x) \cup \overline{B_2}(x)$ in γ. Since x executes $R1$ in $\gamma \rightarrow \gamma'$, $\overline{\alpha} \notin b_x$ in γ'. Hence $\alpha \notin B_4(x)$ in γ'.

From all the above, we can conclude that $B_4(x) = \emptyset$ in γ'. ■

It follows from all the four claims above that $B_1(x) = \emptyset \wedge B_2(x) = \emptyset \wedge B_3(x) = \emptyset \wedge B_4(x) = \emptyset$ in γ'. Thus, x is not privileged in γ' and the lemma is proved. ■

Lemma 9 *If* $\gamma_i \rightarrow \gamma_{i+1}$ *and* $\gamma_j \rightarrow \gamma_{j+1}$ *are two consecutive system moves in* Γ *in which* x *joins, then there exists a child* y *of* x *in* BFS_G *such that* y *joins in a system move in* $(\gamma_i, \ldots, \gamma_j)$*.*

Proof. We prove this claim by contradiction. Suppose no child of x in BFS_G can join in any system move in $(\gamma_i, \ldots, \gamma_j)$. Then no child of x in BFS_G can join in $\gamma_i \rightarrow \gamma_{i+1}$. By Lemma 8, x is not privileged in γ_{i+1} (thus, $i+1 \lneqq j$) and hence $B_1(x) = B_2(x) = B_3(x) = B_4(x) = \emptyset$ in γ_{i+1}. Since x does not join in any system move in $(\gamma_{i+1}, \ldots, \gamma_j)$ and no child of x in BFS_G can join in any system move in $(\gamma_{i+1}, \ldots, \gamma_j)$, the sets b_x, $One(x)$ and $Two(x)$ are all fixed in $(\gamma_{i+1}, \ldots, \gamma_j)$. This combined with the fact that $E(x)$ and $NT(x)$ are fixed in Γ implies that the sets $B_1(x)$, $B_2(x)$, $B_3(x)$ and $B_4(x)$ are all fixed in $(\gamma_{i+1}, \ldots, \gamma_j)$. Hence $B_1(x) = B_2(x) = B_3(x) = B_4(x) = \emptyset$ in γ_j. Thus x is not privileged in γ_j and hence x cannot join in $\gamma_j \rightarrow \gamma_{j+1}$, which causes a contradiction. Thus, the lemma is proved. ■

Lemma 10 *For any node x in the system, if the number of system moves in Γ in which at least one child of x in BFS_G joins is m, then x can join in at most $m+1$ system moves in Γ.*

Proof. Suppose x can join in at least $m+2$ system moves in Γ. Then It follows from Lemma 9 that there exists at least $m+1$ system moves in Γ in which at least one child of x in BFS_G joins, which contradicts the assumption of this claim. Therefore, the claim is proved. ■

Lemma 11 *For any node x in the system, if x has m descendants in BFS_G (note: x is a descendant of itself), then x can join in at most m system moves in Γ.*

Proof. We prove by induction on m.
(1) *Induction basis.* For $m = 1$, let x be any node in the system that has 1 descendant in BFS_G. Then x has no child in BFS_G and hence the number of system moves in which at least one child of x in BFS_G joins is zero. It follows from Lemma 10 that x can join in at most 1 system move in Γ.
(2) *Inductive step.* Assume that $k \geq 1$ and the claim is true for $1 \leq m \leq k$. Let x be any node in the system that has $k+1$ descendants in BFS_G. Let $x_1, x_2, \ldots, x_j$ be all the children of x in BFS_G and $m_{x_1}, m_{x_2}, \ldots, m_{x_j}$ be the number of descendants of $x_1, x_2, \ldots, x_j$ in BFS_G, respectively. Then $m_{x_1}, m_{x_2}, \ldots, m_{x_j}$ are all less than or equal to k. By the induction hypothesis, $x_1, x_2, \ldots, x_j$ can join in at most $m_{x_1}, m_{x_2}, \ldots, m_{x_j}$ system moves in Γ, respectively. Thus, the number of system moves in Γ in which at least one child of x in BFS_G joins is $q \leq m_{x_1} + m_{x_2} + \cdots + m_{x_j} = k$. By Lemma 10, x can join in $q+1 \leq k+1$ system moves in Γ.

By (1), (2) and the postulate of mathematical induction, the lemma is proved. ■

Lemma 12 *If $\Gamma = (\gamma_1, \gamma_2, \ldots)$ is an execution of Algorithm 1, then the length of Γ is at most n^2, where n is the number of nodes in the system.*

Proof. Since any node in the system has at most n descendants in BFS_G, it can join in at most n system moves in Γ by Lemma 11. Thus, the number of system moves in Γ is at most n^2 and the lemma is proved. ■

Theorem 2 *Algorithm 1 is self-stabilizing under the distributed demon model and solves the bridge finding problem. Moreover, Algorithm 1 stabilizes in at most n^2 steps, where n is the number of nodes in the system.*

Proof. This follows from Lemmas 7, 12 and Theorem 1. ■

4 Concluding remarks

In this paper, we have proposed a self-stabilizing algorithm that can find all bridges in a distributed system. Assuming that a BFS tree has been constructed, our algorithm stabilizes under the distributed demon model in at most n^2 steps, where n is the number of nodes in the system. These results are stronger or more general than most of the results in [2], [3] and [5].

Note that the results in this paper as well as the results in [2], [3] and [5] all have the same weakness - each of them depends on an assumption of the presence of some rooted tree (BFS or DFS tree). Although attempts to combine algorithms have been made in [2], [3] and [5] to get rid of such assumptions, all of them are unsuccessful. For any of these attempts, the failure is simply due to the fact that one of the two to-be-combined algorithms is not self-stabilizing under the central demon model. However, it is worth noting that even if the two to-be-combined algorithms are self-stabilizing under the central demon model, the resulting combined algorithm may still not remain self-stabilizing under the central demon model if the way of combining is not appropriate. In other words, the combining of algorithms under the central demon model is not an obvious but a subtle matter that requires a certain degree of carefulness. In our future work, we will clarify the above issue and also find a self-stabilizing bridge-finding algorithm that does not need to assume the presence of any rooted tree.

References

[1] J.E. Burns, "Self-stabilizing ring without demons", Technical Report GIT-ICS-87/36, Georgia Tech., 1987.

[2] P. Chaudhuri, "An $O(n^2)$ self-stabilizing algorithm for computing bridge-connected components", Computing, Vol. 62, pp.55-67, 1999.

[3] S. Devismes, "A silent self-stabilizing algorithm for finding cut-nodes and bridges", Parallel Processing Letters, Vol. 15, pp.183-198, 2005.

[4] E.W. Dijkstra, "Self-stabilizing systems in spite of distributed control", Communcation of ACM, Vol. 17, pp.643-644, 1974.

[5] M. H. Karaata, P. Chaudhuri, "A self-stabilizing algorithm for bridge finding", Distributed Computing, vol. 12, pp.47-53, 1999.