An Efficient Silent Self-Stabilizing Algorithm for 1-Maximal Matching in Anonymous Network

Yuma Asada and Michiko Inoue

Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, NARA 630-0192 JAPAN,
{asada.yuma.ar4, kounoe}@is.naist.jp

Abstract. We propose a new self-stabilizing 1-maximal matching algorithm which is *silent* and works for *anonymous* networks without a cycle of a length of a multiple of 3 under a central *unfair* daemon. Let e be the number of edges and let n be the number of nodes in a graph. The time complexity is $O(e)$ moves. Therefore, the complexity is $O(n)$ moves for trees or rings whose length is not a multiple of 3.

Keywords: distributed algorithm, self-stabilization, graph theory, matching problem

1 Introduction

Self-Stabilization [5] can tolerate several inconsistencies of computer networks caused by transient faults, erroneous initialization, or dynamic topology change. It can recover and stabilize to consistent system configuration without restarting program execution.

Maximum or *maximal matching* is a well-studied fundamental problem for distributed networks. A matching is a set of pairs of adjacent nodes in a network such that any node belongs to at most one pair. It can be used in distributed applications where pairs of nodes, such as a server and a client, are required. This paper proposes an efficient anonymous self-stabilizing algorithm for *1-maximal matching*. A 1-maximal matching is a $\frac{2}{3}$-approximation to the maximum matching, and expected to find more matching than a *maximal matching* which is a $\frac{1}{2}$-approximation to the maximum matching.

Self-stabilizing algorithms for the maximum and maximal matching problems have been well studied[7]. Table 1 summarizes the results, where n and e denote the numbers of nodes and edges, respectively.

Blair and Manne[1] showed that a maximum matching can be solved with $O(n^2)$ moves for tree networks under a read/write daemon. They proposed a leader election algorithm for *non-anonymous* trees where a rooted tree is constructed in a bottom-up fashion, and then showed bottom-up algorithms including a maximum matching[2] can be combined with the proposed leader election algorithm so that the combined algorithm simultaneously solves the two problems. For *anonymous* networks, Karaata et al.[10] proposed a maximum matching algorithm with $O(n^4)$ moves for trees under a central daemon, and Chattopadhyay et al.[3] proposed a maximum matching algorithm with $O(n^2)$ rounds

Table 1. Self-stabilizing matching algorithms.

Reference	Matching	Topology	Anonymous	Daemon	Complexity
[1]	maximum	tree	no	read/write	$O(n^2)$ moves
[10]	maximum	tree	yes	central	$O(n^4)$ moves
[3]	maximum	bipartite	yes	central	$O(n^2)$ rounds
[9]	maximal	arbitrary	yes	central	$O(e)$ moves
[6]	1-maximal	tree, ring	yes	central	$O(n^4)$ moves
[12]	1-maximal	arbitrary	no	distributed	$O(n^2)$ rounds
this paper	1-maximal	arbitrary*	yes	central	$O(e)$ moves

* arbitrary topology without a cycle of length of a multiple of 3.

for bipartite networks under a central daemon. Recently, Datta and Larmore[4] proposed a *silent* weak leader election algorithm for anonymous trees. The algorithm elects one or two co-leaders with $O(n \cdot Diam)$ moves in a bottom-up fashion under an unfair distributed daemon, where $Diam$ is a network diameter. Though there is no description, it seems that it can be combined with the maximum matching algorithm[2] without increasing the time complexity.

Hsu and Huang[9] proposed a maximal matching algorithm for anonymous networks with arbitrary topology under a central daemon. They showed the time complexity of $O(n^3)$ moves, however, it has been revealed that the time complexity of their algorithm is $O(n^2)$ moves by Tel[13] and Kimoto et al.[11] and $O(e)$ moves by Hedetniemi et al. [8].

Goddard et al.[6] proposed a 1-maximal matching with $O(n^4)$ moves for anonymous trees and rings whose length is *not* a multiple of 3 under a central daemon. They also showed that there is no self-stabilizing 1-maximal matching algorithm for anonymous rings with length of a multiple of 3. Manne et al. [12] also proposed a 1-maximal matching algorithm for non-anonymous networks with any topology under a distributed unfair daemon. Their algorithm stabilizes in $O(n^2)$ rounds and $(2^n \cdot \Delta \cdot n)$ moves, where Δ is the maximum degree of nodes.

Our contribution: In this paper, we propose a new self-stabilizing 1-maximal matching algorithm. The proposed algorithm is *silent* and works for *anonymous* networks without a cycle of a length of a multiple of 3 under a central *unfair* daemon. We will show that the time complexity of the proposed algorithm is $O(e)$ moves. Therefore, the complexity is $O(n)$ moves for trees or rings whose length is not a multiple of 3.

The remaining of the paper is organized as follows. In Section 2, we describe a concept of distributed systems, self-stabilization, a computational model and a definition of the 1-maximal matching problem. In Section 3, we explain an overview of our proposed algorithm for 1-maximal matching problem. It includes informal descriptions of variables, states, and behavior in each state. In Section 4, we describe the algorithm. In Section 5, we prove the correctness of our algo-

rithm. The proof is divided into two parts; properties of terminal configuration and time complexity.

2 Preliminaries

A distributed system consists of multiple asynchronous processes. Its topology is represented by an undirected connected graph $G = (V, E)$ where a node in V represents a process and an edge in E represents the interconnection between the processes. A node is a state machine which changes its states by actions. Each node has a set of actions, and a collections of actions of nodes is called a *distributed algorithm*. Let n and e denote the numbers of nodes and edges in a distributed system.

In this paper, we consider *state-reading model* as a communication model where each process can directly read the internal state of its neighbors. An action of a node is expressed $\langle label \rangle :: \langle guard \rangle \mapsto \langle statement \rangle$. A guard is a Boolean function of all the states of the node and its neighbors, and a statement updates its local state. We say a node is privileged if some of its actions has a true guard. Only privileged node can *move* by selecting one action with a true guard and executing its statement.

Moves of processes are scheduled by a *daemon*. Among several daemons considered for distributed systems, we consider an *unfair central daemon* in this paper. A central daemon chooses one privileged node at one time, and the selected node atomically moves. A daemon is unfair in a sense that it can choose any node among privileged nodes.

A problem $\mathcal{P}$ is specified by its legitimate configurations where configuration is a collection of states of all the nodes. We say a distributed algorithm $\mathcal{A}$ is *self-stabilizing* if $\mathcal{A}$ satisfies the following properties. 1) **convergence**: The system eventually reaches to a legitimate configuration from any initial state by distributed algorithm $\mathcal{A}$, and 2) **closure**: The system once reaches to a legitimate configuration, all the succeeding moves keep the system configuration legitimate. A self-stabilizing algorithm is *silent* if, from any arbitrary initial configuration, the system reaches a terminal configuration where no node can move. A self-stabilizing algorithm is *anonymous* if it does not use global IDs of nodes. We only assume that nodes have pointers and a node can determine whether its neighbor points to it.

A *matching* in an undirected graph $G = (V, E)$ is a subset M of E such that each node in V is incident to at most one edge in M. We say a matching is *maximal* if no proper superset of M is a matching as well. A maximal matching M is *1-maximal* if, for any $e \in M$, any matching cannot be produced by removing e from M and adding two edges to $M - \{e\}$. A maximal matching is a $\frac{1}{2}-$approximation to the maximum matching. On the other hand, a 1-maximal matching is a $\frac{2}{3}-$approximation. In this paper, we propose a silent and anonymous self-stabilizing algorithm for the 1-maximal matching problem for graphs without a cycle of length of a multiple of 3.

3 Overview of An Algorithm MM1

First, we will show an overview of a proposed self-stabilizing 1-maximal matching algorithm MM1. Each node i uses stages to construct 1-maximal matching. There are seven stages; *S1a*, *S1b*, *S2a*, *S2b*, *S3*, *S4*, *S5*. Stages *S1a* and *S1b* mean that the node is not matched. A stage *S2a* means the node is matched with a neighbor node. Moreover, *S2b*, *S3*, *S4*, *S5* mean the node is matched and in progress to increase matching. We say that a state of a node is valid if the state represents some stage, and if a node detects its invalidity, the node resets to *S1a*. A node i has three variables; $\texttt{level}_i$, $\texttt{m-ptr}_i$, $\texttt{i-ptr}_i$. We describe how to use the variables in our algorithm.

S1a, S1b, S2a We say a node is *free* if the node is in *S1a* or *S1b*. A node in *S1a* doesn't invite any nodes, while A node in *S1b* invites a node. Fig.1 shows making match behavior of free nodes. When a free node i finds a free neighbor node j, i invites j by $\texttt{i-ptr}_i$. Then invited node j updates its level to 2 and points to i by $\texttt{m-ptr}_j$ to accept the invitation from i. Finally i regards that j accepted i's invitation, and i points to j by m-ptr_i to make match. A node in *S2* is at level 2 and doesn't invite any nodes.

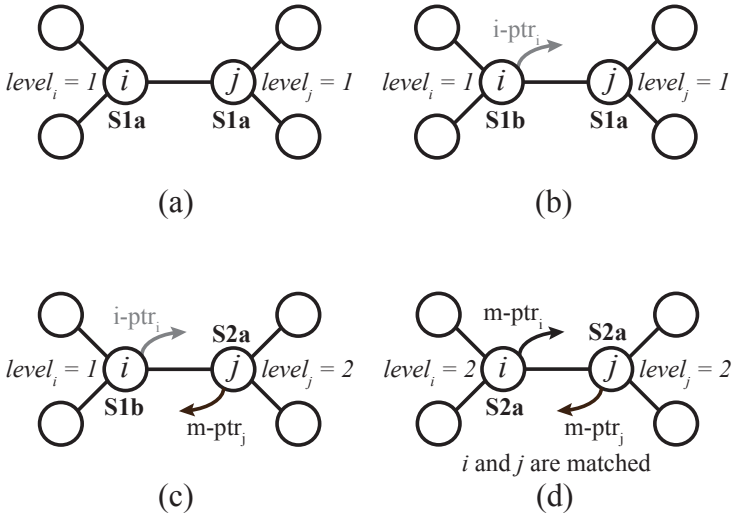

Fig. 1. Making match behavior of free nodes

S2b, S3, S4, S5 Matched nodes try to increase the number of matching M if they have free neighbor nodes. Fig. 2 shows an example increasing match behavior.

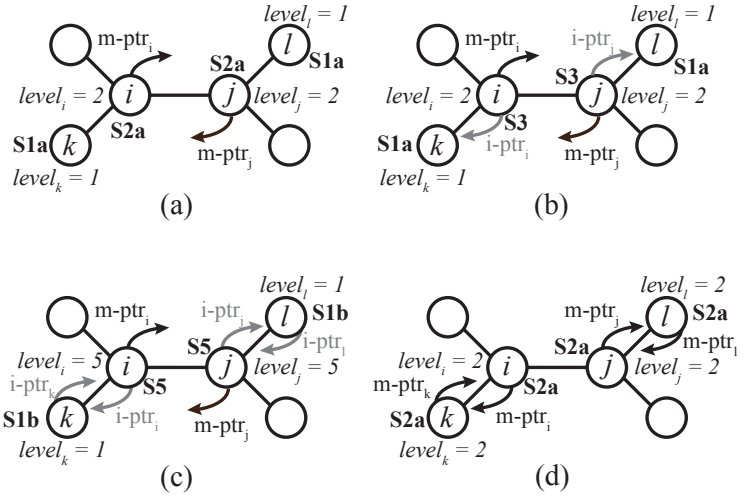

Fig. 2. Increasing matching

In Fig. 2 (a), we can increase matching to break matching between i and j, and create new matchings between i and k, and j and l, respectively. Node i and j invite its free neighbors if exist, and when both nodes i and j invite free neighbor node, they change their level to 3. That indicates that they are ready to be approved as Fig. 2 (b) Then k and l point to the inviting nodes by `i-ptr` to approve their invitations. Node i and j change their level to 4 if the neighbors approve the invitations, and change their level to 5 when they notice that both invitations are approved. This indicates that they are ready to break matching as Fig. 2 (c). Then they create new matchings with the free nodes as Fig. 2 (d).

Reset Each node always check its validity which means three variable level, m-ptr, i-ptr represent some stage. In addition, the validity among the states of the node and its neighbors is also evaluated. For example, if a level is 1 and m-ptr points to some neighbor the state is invalid. If a node is at level 3 and m-ptr_i points to a neighbor j but j's level is 1, a combination of these states is invalid. A node resets to *S1a* if it find its invalidity. In addition, a node does not move while some neighbor is in an invalid state.

Cancel A node cancels an invitation or progress to increase matching, if it detect that the invitation cannot be accepted or it cannot increase matching. When canceling, a node goes back to *S1a* if it is at level 1, to *S2a* if it is at level 2 or more.

4 Algorithm MM1

A node i has three variables, $\texttt{level}_i$, $\texttt{m-ptr}_i$, $\texttt{i-ptr}_i$.

- $\texttt{level}_i \in \{1, 2, 3, 4, 5\}$
- $\texttt{m-ptr}_i \in (N(i) \cap \{\perp\})$
- $\texttt{i-ptr}_i \in (N(i) \cap \{\perp\})$

The validity is defined as follows.

Each stage is represented by some valid combinations of three variables. The predicates in Fig. 3 are used to represent such validities.

S1b_valid(i,k): $\texttt{level}_i = 1 \wedge \texttt{m-ptr}_i = \perp \wedge \texttt{i-ptr}_i = k$
S2a_valid(i,j): $\texttt{level}_i = 2 \wedge \texttt{m-ptr}_i = j \wedge \texttt{m-ptr}_i = \perp$
S2b_valid(i,j,k): $\texttt{level}_i = 2 \wedge \texttt{m-ptr}_i = j \wedge \texttt{m-ptr}_i = k \wedge j \neq k$
S3_valid(i,j,k): $\texttt{level}_i = 3 \wedge \texttt{m-ptr}_i = j \wedge \texttt{m-ptr}_i = k \wedge j \neq k$
S4_valid(i,j,k): $\texttt{level}_i = 4 \wedge \texttt{m-ptr}_i = j \wedge \texttt{m-ptr}_i = k \wedge j \neq k$
S5_valid(i,j,k): $\texttt{level}_i = 4 \wedge \texttt{m-ptr}_i = j \wedge \texttt{m-ptr}_i = k \wedge j \neq k$

Fig. 3. Valid predicates

Fig. 4 shows definitions of one node invalidity of a node which can be evaluated by only a state of the node.

S1a_valid1(i): $\texttt{level}_i = 1 \wedge \texttt{m-ptr}_i = \perp \wedge \texttt{i-ptr}_i = \perp$
S1b_valid1(i): $\exists k \in N(i)$ S1b_valid(i,k)
S2a_valid1(i): $\exists j, k \in N(i)$ S2a_valid(i,j)
S2b_valid1(i): $\exists j, k \in N(i)$ S2b_valid(i,j,k)
S3_valid1(i): $\exists j, k \in N(i)$ S3_valid(i,j,k)
S4_valid1(i): $\exists j, k \in N(i)$ S4_valid(i,j,k)
S5_valid1(i): $\exists j, k \in N(i)$ S4_valid(i,j,k)
valid1(i): *S1a_valid(i)* $\wedge$ *S1b_valid(i)* $\wedge$ *S2a_valid(i)* $\wedge$ *S2b_valid(i)* $\wedge$ *S3_valid(i)* $\wedge$ *S4_valid(i)* $\wedge$ *S5_valid(i)*
invalid1(i): $\neg$ *valid1(i)*

Fig. 4. One node invalid functions

Fig. 5 shows definitions of valid functions that evaluate the validity of stages of a node with states of the node and its neighbors if needed. Fig. 6 shows *statement macros* which are utilized in a statement of action. Fig. 7 shows *guard function* which are utilized in guard of action. Fig. 8 shows a code of the proposed algorithm MM1. Daemon selects the first action in the described order in Fig. 8 if a node satisfies simultaneously some guards.

S1a(i): *S1a_valid1(i)*
S1b(i): *S1b_valid1(i)*
S2a(i): $\exists j \in N(i)(S2a\_valid(i,j) \wedge ((\texttt{level}_j = 2 \vee \texttt{level}_j = 3) \wedge \texttt{m-ptr}_j = i) \vee (\texttt{level}_j = 1 \wedge \texttt{i-ptr}_j = i) \vee (\texttt{level}_j = 5 \wedge \texttt{i-ptr}_j = i))$
S2b(i): $\exists j \in N(i)(S2b\_valid(i,j,k) \wedge (\texttt{level}_j = 2 \vee \texttt{level}_j = 3 \vee \texttt{level}_j = 4) \wedge \texttt{m-ptr}_j = i)$
S3(i): $\exists j \in N(i)(S3\_valid(i,j,k) \wedge (\texttt{level}_j = 2 \vee \texttt{level}_j = 3 \vee \texttt{level}_j = 4) \wedge \texttt{m-ptr}_j = i)$
S4(i): $\exists j,k \in N(i)(S4\_valid(i,j,k) \wedge (\texttt{level}_j = 2 \vee \texttt{level}_j = 3 \vee \texttt{level}_j = 4 \vee \texttt{level}_j = 5) \wedge \texttt{m-ptr}_j = i \wedge \texttt{level}_k = 1 \wedge \texttt{i-ptr}_k = i \wedge \texttt{i-ptr}_j \neq \perp)$
S5(i): $\exists j,k \in N(i)(S5\_valid(i,j,k) \wedge (\texttt{level}_k = 1 \wedge \texttt{i-ptr}_k = i) \vee (\texttt{level}_k = 2 \wedge \texttt{m-ptr}_k = i))$
valid(i): *S1a(i)* $\wedge$ *S1b(i)* $\wedge$ *S2a(i)* $\wedge$ *S2b(i)* $\wedge$ *S3(i)* $\wedge$ *S4(i)* $\wedge$ *S5(i)*
invalid(i): $\neg$ *valid(i)*

Fig. 5. Valid functions

`make_match`: $\texttt{i-ptr}_i = \perp, \texttt{m-ptr}_i = j, \texttt{level}_i = 2$
`reset_state`: $\texttt{i-ptr}_i = \perp, \texttt{m-ptr}_i = \perp, \texttt{level}_i = 1$
`abort_exchange`: $\texttt{i-ptr}_i = \perp, \texttt{level}_i = 2$

Fig. 6. Statement macros

no_invalid1_neighbor(i): $\forall x \in N(i)$ *valid1(x)*

Fig. 7. Guard function

5 Correctness

5.1 Correctness in Properties on Terminal Configuration

Lemma 1. *There are no nodes whose level is 5 in any terminal configuration of MM1.*

Proof. By contradiction. Assume that there is a node i that is in *S5* in a terminal configuration. Therefore, i satisfies $\texttt{m-ptr}_i \neq \perp$ and $\texttt{i-ptr}_i \neq \perp$. Let k be a node pointed by $\texttt{i-ptr}_i$. Node k satisfies $\texttt{level}_k = 1 \wedge \texttt{i-ptr}_k = i$ or $\texttt{level}_k = 2 \wedge \texttt{m-ptr}_k = i$ because i satisfies *S5(i)*. If it is $\texttt{level}_k = 1$, k can execute `migrate1`. Also i can execute `migrate2` if it is $\texttt{level}_k = 2$. A contradiction. □

Lemma 2. *A node that points to its neighbor node by* `m-ptr` *also pointed by the neighbor's* `m-ptr` *in any terminal configuration of MM1.*

Proof. By contradiction. There are no node whose level is 5 in any terminal configuration and all nodes are valid. Assume that there are nodes i and i's neighbor $j \in N(i)$ and $\texttt{m-ptr}_i = j \wedge \texttt{m-ptr}_j \neq i$. A node i is in *S2a* since i cannot satisfy *S2b*, *S3* and *S4*. A node j is at level 1 and points to i by $\texttt{i-ptr}_j$ because i satisfies *S2a*. Since i is in *S2a* and j is $\texttt{level}_j = 1 \wedge \texttt{i-ptr}_j = i$, j can execute `match3`. A contradiction. □

Reset
reset1 :: *invalid1(i)* $\mapsto$ reset_state
reset2 :: *invalid1(i)* $\wedge$ *no_invalid1_neighbor(i)* $\mapsto$ reset_state

S1a
match1 :: *S1a(i)* $\wedge$ *no_invalid1_neighbor(i)* $\wedge \exists x \in N(i)(\texttt{i-ptr}_x = i \wedge \texttt{level}_x = 1) \mapsto \texttt{i-ptr}_i = \perp, \texttt{m-ptr}_i = x, \texttt{level}_i = 2$
approve1 :: *S1a(i)* $\wedge$ *no_invalid1_neighbor(i)* $\wedge \exists x \in N(i)(\texttt{i-ptr}_x = i \wedge \texttt{level}_x = 3) \mapsto \texttt{i-ptr}_i = x$
invite1 :: *S1a(i)* $\wedge$ *no_invalid1_neighbor(i)* $\wedge \exists x \in N(i) \texttt{level}_x = 1 \mapsto \texttt{i-ptr}_i = x$

S1b
match2 :: *S1b(i)* $\wedge$ *no_invalid1_neighbor(i)* $\wedge (\exists x \in N(i)(\texttt{i-ptr}_x = i \wedge \texttt{level}_x = 1)) \wedge (\exists k \in N(i)(S1b\_valid(i,k) \wedge \texttt{level}_k < 4)) \mapsto \texttt{i-ptr}_i = \perp, \texttt{m-ptr}_i = x, \texttt{level}_i = 2$
match3 :: *S1b(i)* $\wedge$ *no_invalid1_neighbor(i)* $\wedge (\exists k \in N(i)(S1b\_valid(i,k) \wedge \texttt{m-ptr}_k = i \wedge \texttt{level}_k = 2)) \mapsto$ make_match
migrate1 :: *S1b(i)* $\wedge$ *no_invalid1_neighbor(i)* $\wedge (\exists k \in N(i)(S1b\_valid(i,k) \wedge \texttt{i-ptr}_k = i \wedge \texttt{level}_k = 5)) \mapsto$ make_match
cancel1 :: *S1b(i)* $\wedge$ *no_invalid1_neighbor(i)* $\wedge (\exists k \in N(i)(S1b\_valid(i,k) \wedge (\texttt{m-ptr}_k \neq i \wedge \texttt{level}_k = 5) \vee (\texttt{i-ptr}_k \neq i \wedge \texttt{level}_k \geq 3))) \mapsto$ make_match

S2a
invite2 :: *S2a(i)* $\wedge$ *no_invalid1_neighbor(i)* $\wedge (\exists x \in N(i) \texttt{level}_x = 1) \wedge (\exists j \in N(i)(S2a\_valid(i,j) \wedge \texttt{m-ptr}_j = i)) \mapsto \texttt{i-ptr}_i = x$

S2b
cancel2 :: *S2b(i)* $\wedge$ *no_invalid1_neighbor(i)* $(\exists j, k \in N(i)(S2b\_valid(i,j,k) \wedge \texttt{level}_k \geq 2)) \mapsto$ abort_exchange
proceed1 :: *S2b(i)* $\wedge$ *no_invalid1_neighbor(i)* $(\exists j, k \in N(i)(S2b\_valid(i,j,k) \wedge \texttt{i-ptr}_j \neq \perp)) \mapsto \texttt{level}_i = 3$

S3
cancel3 :: *S3(i)* $\wedge$ *no_invalid1_neighbor(i)* $(\exists j \in N(i)(S3\_valid(i,j,k) \wedge (\texttt{level}_j = 2 \wedge \texttt{i-ptr}_j = \perp) \vee \texttt{level}_k \neq 1) \mapsto$ abort_exchange
proceed2 :: *S3(i)* $\wedge$ *no_invalid1_neighbor(i)* $(\exists j, k \in N(i)(S3\_valid(i,j,k) \wedge \texttt{i-ptr}_k = i \wedge \texttt{level}_k = 1)) \mapsto \texttt{level}_i = 4$

S4
proceed3 :: *S4(i)* $\wedge$ *no_invalid1_neighbor(i)* $(\exists j \in N(i)(S4\_valid(i,j,k) \wedge (\texttt{level}_j = 4 \vee \texttt{level}_j = 5))) \mapsto \texttt{level}_i = 5$

S5
migrate2 :: *S5(i)* $\wedge$ *no_invalid1_neighbor(i)* $(\exists j, k \in N(i)(S5\_valid(i,j,k) \wedge \texttt{level}_k = 2 \wedge \texttt{m-ptr}_k = i)) \mapsto \texttt{i-ptr}_i = \perp, \texttt{m-ptr}_i = k, \texttt{level}_i = 2$

Fig. 8. Algorithm MM1

Lemma 3. *There are no two nodes i and j such that $\texttt{level}_i = 1$, $\texttt{level}_j = 3$ or 4, $\texttt{i-ptr}_i = j$ and $\texttt{i-ptr}_j = i$ in any termination configuration of MM1 for any graphs without a cycle of length of a multiple of 3.*

Proof. By contradiction. There are no nodes whose level is 5 in any terminal configuration and all nodes are valid. Assume that there are adjacent nodes i which is at 1 level and $\texttt{i-ptr}_i = j$, and j which is at level 3 or 4 and $\texttt{i-ptr}_j = i$. If i and j satisfy $\texttt{level}_i = 1 \wedge \texttt{level}_j = 3$, j can execute `proceed2` since j is in *S3*.

Consider the case where i and j satisfy $\texttt{level}_i = 1 \wedge \texttt{level}_j = 4$. There is a node $k \in N(j)$ such that $\texttt{level}_k = 2$ or 3 or 4, $\texttt{m-ptr}_j = k, \texttt{i-ptr}_k \neq \perp$. Moreover, k can execute `proceed1` if $\texttt{level}_k = 2$ and also j can execute `proceed3` if $\texttt{level}_k = 4$. Hence $\texttt{level}_k$ is limited to 3. Therefore, there is a node $l \in N(k)$ such that $\texttt{i-ptr}_k = l$ and $\texttt{level}_l = 1$ since k is in *S3*. Node l satisfies $\texttt{i-ptr}_l \neq k$ because it is in termination configuration. Furthermore, there is a node $m \in N(l)$ such that $\texttt{i-ptr}_l = m$ and $\texttt{level}_m = 4$. In addition, there is a node $o \in N(o)$ which is at level 3 in the same reason above.

Repeating the above observation, we can show there is an infinity sequence of nodes $1, 4, 3, 1, 4, 3, \cdots$. However there is no such sequence since there is no cycle of length of a multiple of 3. A contradiction. □

Theorem 1. *A maximal matching is constructed in any terminal configuration of MM1 for any graphs without a cycle of length of a multiple of 3.*

Proof. By contradiction. There are no nodes whose level is 5 in any terminal configuration and all nodes are valid. Assume that a matching is not maximal in some terminal configuration. Hence there are adjacent nodes i and j such that their level is 1 by the assumption and Lemma 2.

If node i or j is in *S1a*, it can execute `invite1` (Observation 1).

Consider the case where both i and j are in *S1b*. Let node k be pointed by $\texttt{i-ptr}_i$. The level of node k is less than 5 by Lemma 1.

Consider the case where $\texttt{level}_k = 1$. A node k is in *S1b* by the same reason as Observation 1. Let x a node pointed by $\texttt{i-ptr}_k$. A node k can execute `match2` if $\texttt{level}_x \neq 4$. It satisfies $\texttt{i-ptr}_x = k$ if $\texttt{level}_x = 4$ by Lemma 3. Therefore, k can execute `cancel1`. Consider the case where $\texttt{level}_k = 2$. Node k is in *S2b* because k can execute `invite2` if k is in *S2a*. Then i can execute `cancel1` since $\texttt{m-ptr}_k \neq i$. If it satisfies $\texttt{level}_k = 3$, i can execute `cancel1` by Lemma 3 because $\texttt{i-ptr}_k \neq i$. If it satisfies $\texttt{level}_k = 4$, i can execute `cancel1` as in the case of $\texttt{level}_k = 3$. A contradiction. □

Theorem 2. *A 1-maximal matching is constructed in any terminal configuration of the algorithm MM1 for any graphs without a cycle of length of a multiple of 3.*

Proof. By contradiction. Assume that a matching is not 1-maximal in some terminal configuration. Since it is terminal, a maximal matching is constructed by Theorem 1. Therefore, there are matching nodes i and j and both have

neighbors at level 1 since any node at level 2 or more has matching neighbor by Lemma 2.

Both i and j are at level 2 or more since they are matching, and any of them is not in $S1a$ since they have level 1 neighbor and can execute `invite1` if they are in $S1a$, or not at level 5 by Lemma 1. Since i and j are in $S2b$, $S3$ or $S4$, both nodes point to some neighbor by `i-ptr`, and the neighbors are at level 1. That is because, i or j can execute `cancel2` in $S2b$, `cancel3` in $S3$ and `reset2` in $S4$ if it points to a node at level 2 or more.

Nodes i and j are not in $S2b$ since $\texttt{i-ptr}_i \neq \perp$ and $\texttt{i-ptr}_j \neq \perp$, and therefore, they can execute `proceed1` if they are in $S2b$.

Consider the case where i or j is in $S3$. Assume i is in $S3$ w.o.l.g., and let k be a level 1 node that i points to by `i-ptr`. A node k can execute `approve` if $\texttt{i-ptr}_k \neq \perp$, and node i can execute `proceed2` if $\texttt{i-ptr}_k \neq i$. Therefore, $\texttt{i-ptr}_k \neq x$ for some $x \neq i$. Since there is no adjacent level 1 nodes by Theorem 1, there is no level 5 node by Lemma1, and `m-ptr`s point to each other between two matching nodes by Lemma 2, x is at level 2, 3, or 4, and $\texttt{m-ptr}_x \neq k$. A node x is not at level 2 since k can execute `cancel1` if x is at level 2. In case where x is at level 3 or 4, $\texttt{i-ptr}_x \neq k$ by Lemma 3, and therefore, k can also execute `cancel1`. Therefore, none of i and j is not in $S3$.

That is, both i and j are in $S4$, however, both can execute `proceed3` in this case. A contradiction. □

5.2 Termination and Time Complexity

Lemma 4. *If a node i at level 1 is valid, that is S1a(i) or S1b(i) holds, i is valid while it is at level 1 in MM1.*

Proof. Validity functions *S1a(i)* and *S1b(i)* checks only the variables of a node i. That is the validity of a node at level 1 is independent of its neighbors' states. Any move for *S1a* or *S1b* keeps the state of node valid, a valid node at level 1 is valid while it is at level 1. □

Lemma 5. *Once a node executes one of* `match1`*,* `match2`*,* `match3`*,* `migrate1` *and* `migrate2`*, the node never reset in MM1.*

Proof. By contradiction. Assume some nodes reset after executing `match1`, `match2`, `match3`, `migrate1` or `migrate2`. Let i be a node that executes such a reset r first. Let m be the last move among `match1`, `match2`, `match3`, `migrate1` and `migrate2` before the reset. Since no move except `reset1` and `reset2` brings invalid states and i already executed m, when i executes r, i is two node invalid. Therefore, i detects some invalidity between i and some neighbor.

Let k be a node such that $\texttt{i-ptr}_i = k$ when i executes r. From the algorithm, that is when i is at level 4 or 5. When i moves to *S4* by `proceed2`, i confirms that k's validity, $\texttt{level}_k = 1$ and $\texttt{i-ptr}_k = i$. Node k never reset while it is at level 1 by Lemma 4 and the validity between i and k is preserved. Node k may move to *S1b* by `migrate1` but never reset before r by the assumption, and therefore, the validity i and k is also preserved.

Therefore, i executes r by detecting invalidity between i and j such that $\texttt{m-ptr}_i = j$. Since m is the last chance to set `m-ptr` for i, i sets $\texttt{m-ptr}_i = j$ by m. When i executes m, j is in *S1b*, *S2a*, or *S5*.

In case of *S1b*, when i executes m, i confirms j's validity and $\texttt{i-ptr}_j = i$. Node j is valid while it is at level 1 by Lemma4. Node i moves to *S2b* after j sets $\texttt{m-ptr}_j = i$ by `match2` or `match3` and moves to *S2a*. Therefore, while j is at level 1, $\texttt{i-ptr}_j = i$ always holds and therefore i cannot reset. After j moves to level 2 by `match2` or `match3`, j does not reset before r from the assumption. Therefore, the validity between i and j is preserved until r.

In case of *S5*, when i executes m, i confirms $\texttt{i-ptr}_j = i$. Since the validity of a node in *S5* only depends on its state and a state of a node pointing to by `i-ptr`, j is valid if the validity between i and j is preserved. Since i does not reset between m and r, the validity is preserved while j is in *S5*. After j moves to level 2 by `migrate2`, j does not reset before r from the assumption. Therefore, the validity between i and j is preserved until r.

In case of *S2a*, i confirms the validity between i and j and $\texttt{m-ptr}_j = i$ when i executes m. Since j is in *S2a*, $\texttt{i-ptr}_j$ does not points to any node. Therefore, even if j points to some node by `i-ptr` after m, the validity between j and the pointed node is preserved like between i and k. Therefore j is valid if the validity between i and j is preserved while $\texttt{m-ptr}_j = i$ and $\texttt{level}_j \leq 4$ (When j moves to *S5*, it does not take care of i). Since i does not reset between m and r, the validity is preserved. □

Lemma 6. *Each node resets at most once in MM1.*

Proof. Once a node execute `reset1` or `reset2`, it moves to *S1a*. The node never reset while it is at level 1 from Lemma 4. The node executes `match1`, `match2` or match3 to moves to level 2, and never reset after that by Lemma 5. □

Lemma 7. *In MM1,* `cancel1`*,* `cancel2` *and* `cancel3` *are executed* $O(e)$ *times.*

We say a move is *progress move* if it is `match1`, `match2`, `match3`, or `migrate1`. A level of node changes from 1 to 2 by a progress move.

Lemma 8. *Progress moves are executed* $O(n)$ *times in MM1.*

Proof. Only `reset1` and `reset2` change a level of a node from 2 or more to 1. Each node executes these reset actions at most once by Lemma 6. Hence, progress moves are executed $O(n)$ times. □

Lemma 9. *In MM1,* `migrate2` *is executed* $O(n)$ *times.*

Theorem 3. *MM1 is silent and takes* $O(e)$ *moves to construct 1-maximal matching for any graphs without a cycle of length of a multiple of 3.*

Proof. Fig. 9 shows each stage transition in MM1. In MM1, each node moves to a higher stage from the current stage in the order of $S1a$, $S1b$, $S2a$, $S2b$, $S3$, $S1a$, $S4$ and $S5$ except `reset1`, `reset2`, `cancel1`, `cancel2`, `cancel2`, `cancel3`

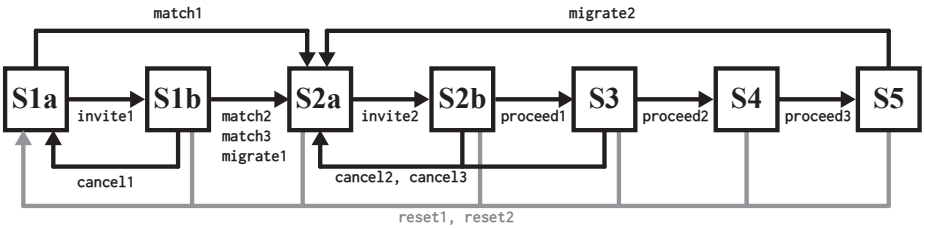

Fig. 9. Transitions of stages

and `migrate2`. Therefore, if a node does not execute these moves, the number of executed moves is at most 6.

Let R_i, C_i and M_i be the numbers of resets (`reset1` or `reset2`), cancels (`cancel1`, `cancel2`, `cancel2` or `cancel3`), and `migrate2` executed by a node i. Let MOV_i denote the total number of moves executed by a node i. From the observation, it is bounded as follows.

$$MOV_i \leq 6(R_i + C_i + M_i + 1)$$

From Lemmas 6, 7 and 9, we have

$$\Sigma_{i \in V} R_i = O(n), \Sigma_{i \in V} C_i = O(e), \text{and } \Sigma_{i \in V} M_i = O(n).$$

Therefore, the total number of moves executed in MM1 can be derived as follows.

$$\Sigma_{i \in V} MOV_i \leq 6(\Sigma_{i \in V} R_i + \Sigma_{i \in V} C_i + \Sigma_{i \in V} M_i) = O(e)$$

Since each node always executes a finite number of moves, MM1 always reaches a terminal configuration where 1-maximal matching is constructed by Theorem 2. This also implies MM1 is silent. □

6 Conclusion

We proposed a 1-maximal matching algorithm MM1 that is silent and works for anonymous networks without a cycle of a length of a multiple of 3 under a central unfair daemon. The time complexity of MM1 is $O(e)$ moves. Therefore, it is $O(n)$ moves for trees or rings whose length is not a multiple of 3. We had a significant improvement from Goddarda et al.[6] that is also anonymous 1-maximal matching algorithm but work for only trees or rings which length is not a multiple of 3 and the time complexity is $O(n^4)$.

References

1. Jean RS Blair and Fredrik Manne. Efficient self-stabilizing algorithms for tree networks. In *Proceedings. 23rd International Conference on Distributed Computing Systems*, pages 20–26. IEEE, 2003.

2. JRS Blair, SM Hedetniemi, ST Hedetniemi, and DP Jacobs. Self-stabilizing maximum matchings. *Congressus Numerantium*, pages 151–160, 2001.
3. Subhendu Chattopadhyay, Lisa Higham, and Karen Seyffarth. Dynamic and self-stabilizing distributed matching. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 290–297. ACM, 2002.
4. Ajoy K Datta and Lawrence L Larmore. Leader election and centers and medians in tree networks. In *Stabilization, Safety, and Security of Distributed Systems*, pages 113–132. Springer, 2013.
5. Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, November 1974.
6. Wayne Goddard, Stephen T Hedetniemi, Zhengnan Shi, et al. An anonymous self-stabilizing algorithm for 1-maximal matching in trees. In *Proceedings International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 797–803, 2006.
7. Nabil Guellati and Hamamache Kheddouci. A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs. *Journal of Parallel and Distributed Computing*, 70(4):406–415, 2010.
8. Stephen T Hedetniemi, David P Jacobs, and Pradip K Srimani. Maximal matching stabilizes in time $O(m)$. *Information Processing Letters*, 80(5):221–223, 2001.
9. Su-Chu Hsu and Shing-Tsaan Huang. A self-stabilizing algorithm for maximal matching. *Information Processing Letters*, 43(2):77–81, 1992.
10. Mehmet Hakan Karaata and Kassem Afif Saleh. Distributed self-stabilizing algorithm for finding maximum matching. *Comput Syst Sci Eng*, 15(3):175–180, 2000.
11. Masahiro Kimoto, Tatsuhiro Tsuchiya, and Tohru Kikuno. The time complexity of Hsu and Huang's self-stabilizing maximal matching algorithm. *IEEE Trans. Infrmation and Systems*, E93-D(10):2850–2853, 2010.
12. Fredrik Manne, Morten Mjelde, Laurence Pilard, and Sébastien Tixeuil. A self-stabilizing 2/3-approximation algorithm for the maximum matching problem. *Theoretical Computer Science*, 412(40):5515–5526, 2011.
13. Gerard Tel. *Introduction to distributed algorithms*. Cambridge university press, 2000.