A Silent Self-Stabilizing Algorithm for 1-Maximal Matching in Anonymous Networks

Yuma Asada and Michiko Inoue

Nara Institute of Science and Technology, 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 any *anonymous* networks without a cycle of a length of a multiple of 3 under a central *unfair* daemon. Let n and e be the numbers of nodes and edges in a graph, respectively. 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. That is a significant improvement from the best existing results of $O(n^4)$ moves for the same problem setting.

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 pairs 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 non-anonymous tree networks under a read/write daemon. They proposed an algorithm to construct a rooted tree, and showed bottom-up algorithms including a maximum matching[2] can be combined with the proposed algorithm so that the combined algorithm simultaneously solves the two problems without increasing the time complexity. For *anonymous* networks, Karaata et

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

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

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 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, and, 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 $O(2^n \cdot \Delta \cdot n)$ 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 any *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. That is a significant improvement of the best existing result of $O(n^4)$ for the same problem setting[6].

The remaining of the paper is organized as follows. In Section 2, we define distributed systems and the 1-maximal matching problem. A 1-maximal matching algorithm is proposed in Section 3, and proves for its correctness and performance are given in Section 4. Finally Section 5 concludes this paper.

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 collection 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 node 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 it has an action with a true guard. Only privileged node can *move* by selecting one action with a true guard and executing its statement.

Moves of nodes 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, 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 itself, some other nodes, or no node.

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 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*, and *S5*. Stages *S1a* and *S1b* mean that the node is not matching with any node. A stage *S2a* means the node is matching with a neighbor node, and, *S2b*, *S3*, *S4*, *S5* mean the node is trying to increase matches. 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* does not invite any nodes, while a node in *S1b* invites its neighbor node. Fig.1 shows how free nodes make a match. When a free node i finds a free neighbor node j, i invites j by $\texttt{i-ptr}_i$ (i is in *S1b*). Then invited node j updates its level to 2 and points to i by $\texttt{m-ptr}_j$ to accept the invitation (j is in *S2a*). Finally i points to j by $\texttt{m-ptr}_i$ to make a match (i is in *S2a*). A node in *S2a* is at level 2 and does not invite any nodes. If two adjacent nodes i and j point to each other by `m-ptr`, we consider they are matching, that is $(i, j) \in M$.

S2b, S3, S4, S5 Matching nodes try to increase the number of matches if they have free neighbor nodes. Fig.2 shows how to increase matches, where matches are increased by breaking a match between i and j, and creating new matches between i and k, and j and l. In Fig.2(a), nodes i and j invite their free neighbors k and l if they do not invite i and j, respectively (i and j are in *S2b*). When both nodes notice that i and j invite free neighbor nodes, they change their level to 3 (i and j are in *S3*). That indicates that they are ready to be approved as in Fig.2(b). Then k and l point to the inviting nodes by `i-ptr` to approve their invitations (k and l are in *S1b*). Node i and j change their level to 4 if the neighbors approve the invitations (i and j are in *S4*) as in Fig.2(c), and change their level to 5 when they notice that both invitations are approved (i and j are in *S5*). This indicates that they are ready to break a match as in Fig.2(d). Then they create new matches with the free nodes, where k and l first move to *S2a* (Fig.2(e)) and then i and j move to *S2a* (Fig.2(f)), respectively. A node in *S1a* or *S1b* can make a match with the other node while an inviting node is in *S3*. However, once the inviting node moves to *S4*, it cannot change its `i-ptr` while the inviting node is in *S4*.

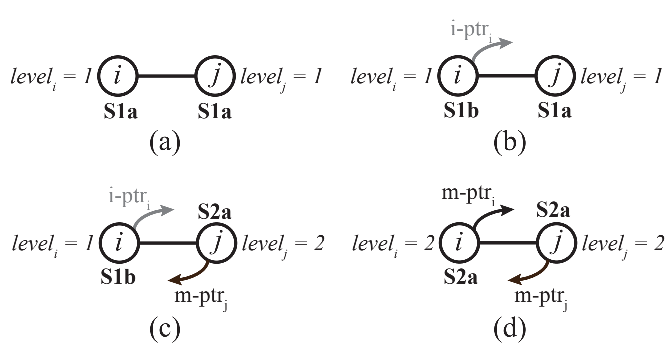

Fig. 1. Making a match between free nodes

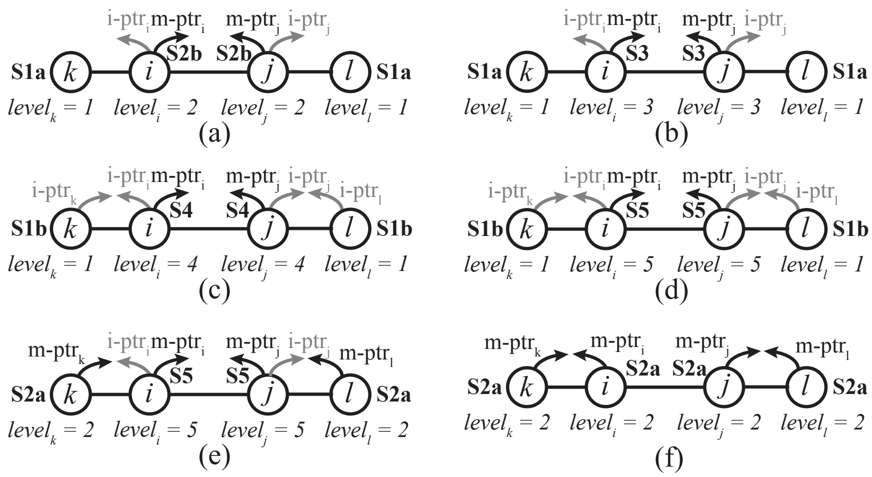

Fig. 2. Increasing matches

Reset Each node always checks its validity, and resets to *S1a* if it finds its invalidity. We consider two kinds of validities, *one node validity* and *two nodes validity*. The one node validity means that a state represents some stage. For example, if a level is 1 and `m-ptr` points to some neighbor, the state is one node invalid. The two nodes validity means that a relation between states of two adjacent nodes is consistent. For example, if a node i is in *S2a*, a node pointed to by `m-ptr` should point to i by `m-ptr` at level 2 or by `i-ptr` at level 1 or 5. The full definition of the validity function is shown in Fig.3. A node does not move while some neighbor is one node invalid.

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

The algorithm MM1 uses some statement macros and a guard function. The variables, validity functions, statement macros and a guard function are shown in Fig.3, and a code of MM1is shown in Fig.4. In the algorithm, each node i uses $N(i)$ to represent a set of its neighbors. That is a set of local IDs for each node and the algorithm does not use any global IDs. We only assume that each node can determine whether its neighbor point to itself, some other node, or no node by pointers `i-ptr` and `m-ptr`.

4 Correctness

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

Proof. By contradiction. Assume that a node i is in *S5* in a terminal configuration. In this case, $\texttt{i-ptr}_i = k$ holds for some k, and $\texttt{level}_k = 1 \wedge \texttt{i-ptr}_k = i$ or

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

Valid Predicates
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): $\texttt{level}_i = 4 \wedge \texttt{m-ptr}_i = j \wedge \texttt{m-ptr}_i = k \wedge j \neq k$

One Node Validity
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)*

Valid Functions (One Node Validity and Two Node Validity)
S1a(i): *S1a_valid1(i)*
S1b(i): *S1b_valid1(i)*
S2a(i): $\exists j \in N(i)($*S2a_valid(i,j)* $\wedge (\texttt{level}_j = 2 \wedge \texttt{m-ptr}_j = i) \vee (\texttt{level}_j = 1 \wedge \texttt{i-ptr}_j = i) \vee (\texttt{level}_j = 5 \wedge \texttt{i-ptr}_j = i))$
S2b(i): $\exists j, k \in N(i)($*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): $\exists j, k \in N(i)($*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)* $\wedge (\texttt{level}_j = 2 \vee \texttt{level}_j = 3 \vee \texttt{level}_j = 4 \vee \texttt{level}_j = 5) \wedge \texttt{m-ptr}_j = i \wedge \texttt{i-ptr}_j \neq \perp \wedge \texttt{level}_k = 1 \wedge \texttt{i-ptr}_k = i)$
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)*

Statement Macros
`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`: $\texttt{i-ptr}_i = \perp, \texttt{level}_i = 2$

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

Fig. 3. Variables, validity functions, statement macros and guard function

Reset

$\texttt{reset1}$ :: *invalid1(i)* $\mapsto \texttt{reset\_state}$

$\texttt{reset2}$ :: *invalid1(i)* $\wedge$ *no_invalid1_neighbor(i)* $\mapsto \texttt{reset\_state}$

S1a

$\texttt{match1}$ :: $S1a(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists x \in N(i)(\texttt{i-ptr}_x = i \wedge \texttt{level}_x = 1) \mapsto \texttt{i-ptr}_i = \perp, \texttt{m-ptr}_i = x, \texttt{level}_i = 2$

$\texttt{approve1}$ :: $S1a(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists x \in N(i)(\texttt{i-ptr}_x = i \wedge \texttt{level}_x = 3) \mapsto \texttt{i-ptr}_i = x$

$\texttt{invite1}$ :: $S1a(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists x \in N(i)\texttt{level}_x = 1 \mapsto \texttt{i-ptr}_i = x$

S1b

$\texttt{match2}$ :: $S1b(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists x \in N(i)(\texttt{i-ptr}_x = i \wedge \texttt{level}_x = 1) \wedge \exists k \in N(i)(S1b\_valid(i,k) \wedge \texttt{level}_k < 4) \mapsto \texttt{i-ptr}_i = \perp, \texttt{m-ptr}_i = x, \texttt{level}_i = 2$

$\texttt{match3}$ :: $S1b(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists k \in N(i)(S1b\_valid(i,k) \wedge \texttt{m-ptr}_k = i \wedge \texttt{level}_k = 2) \mapsto \texttt{make\_match}$

$\texttt{migrate1}$ :: $S1b(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists k \in N(i)(S1b\_valid(i,k) \wedge \texttt{i-ptr}_k = i \wedge \texttt{level}_k = 5) \mapsto \texttt{make\_match}$

$\texttt{cancel1}$ :: $S1b(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists k \in N(i)(S1b\_valid(i,k) \wedge (\texttt{level}_k = 2 \vee (\texttt{level}_k = 3 \wedge \texttt{i-ptr}_k \neq i) \vee (\texttt{level}_k = 4 \wedge \texttt{i-ptr}_k \neq i) \vee (\texttt{level}_k = 5 \wedge \texttt{i-ptr}_k \neq i))) \mapsto \texttt{i-ptr}_i = \perp$

S2a

$\texttt{invite2}$ :: $S2a(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists x \in N(i)(\texttt{level}_x = 1 \wedge \texttt{i-ptr}_x \neq i) \wedge \exists j \in N(i)(S2a\_valid(i,j) \wedge \texttt{m-ptr}_j = i) \mapsto \texttt{i-ptr}_i = x$

S2b

$\texttt{cancel2}$ :: $S2b(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists j,k \in N(i)(S2b\_valid(i,j,k) \wedge \texttt{level}_k \geq 2) \mapsto \texttt{abort\_exchange}$

$\texttt{proceed1}$ :: $S2b(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists j,k \in N(i)(S2b\_valid(i,j,k) \wedge \texttt{i-ptr}_j \neq \perp) \mapsto \texttt{level}_i = 3$

S3

$\texttt{cancel3}$ :: $S3(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists j,k \in N(i)(S3\_valid(i,j,k) \wedge ((\texttt{level}_j = 2 \wedge \texttt{i-ptr}_j = \perp) \vee \texttt{level}_k \geq 2)) \mapsto \texttt{abort\_exchange}$

$\texttt{proceed2}$ :: $S3(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists j,k \in N(i)(S3\_valid(i,j,k) \wedge \texttt{i-ptr}_k = i \wedge \texttt{level}_k = 1) \mapsto \texttt{level}_i = 4$

S4

$\texttt{cancel4}$ :: $S4(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists j,k \in N(i)(S4\_valid(i,j,k) \wedge \texttt{level}_j = 2 \wedge \texttt{i-ptr}_j = \perp) \mapsto \texttt{abort\_exchange}$

$\texttt{proceed3}$ :: $S4(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists j,k \in N(i)(S4\_valid(i,j,k) \wedge (\texttt{level}_j = 4 \vee \texttt{level}_j = 5)) \mapsto \texttt{level}_i = 5$

S5

$\texttt{migrate2}$ :: $S5(i) \wedge no\_invalid1\_neighbor(i) \wedge \exists j,k \in N(i)(S5\_valid(i,j,k) \wedge \texttt{level}_k = 2 \wedge \texttt{m-ptr}_k = i \wedge \texttt{i-ptr}_k = \perp \wedge \texttt{level}_j = 5) \mapsto \texttt{i-ptr}_i = \perp, \texttt{m-ptr}_i = k, \texttt{level}_i = 2$

Fig. 4. Algorithm MM1

$\texttt{level}_k = 2 \wedge \texttt{m-ptr}_k = i$ holds since i is in *S5*. If it is $\texttt{level}_k = 1$, k can execute `migrate1`. If it is $\texttt{level}_k = 2$, i can execute `migrate2`. 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 is no node at level 5 in any terminal configuration and all nodes are valid. Assume that there are adjacent nodes i and j such that $\texttt{m-ptr}_i = j \wedge \texttt{m-ptr}_j \neq i$. A node i is in *S2a* since validity *S2b*(i), *S3*(i) or *S4*(i) do not hold. A node j is at level 1 and $\texttt{i-ptr}_j = i$ from *S2a*(i). Since i is in *S2a* and j is $\texttt{level}_j = 1 \wedge \texttt{i-ptr}_j = i$, j can execute `match3`. A contradiction. □

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

Consider the case of $\texttt{level}_j = 4$. There is a node $k \in N(j)$ such that $\texttt{level}_k = 2$ or 3 or 4, $\texttt{m-ptr}_j = k, \texttt{i-ptr}_k \neq \perp$. Node k can execute `proceed1` if $\texttt{level}_k = 2$ and 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 $\texttt{level}_l = 1$. Node l satisfies $\texttt{i-ptr}_l \neq k$ because it is in a terminal configuration. Therefore, there is a node $m \in N(l)$ such that $\texttt{i-ptr}_l = m$ and $\texttt{level}_m = 4$. Repeating the above observation, we can show there is an infinite sequence of nodes at levels $1, 4, 3, 1, 4, 3, \cdots$. However, there is no such a 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 is no node at level 5 in any terminal configuration and all nodes are valid. Assume that a matching is not maximal in some terminal configuration. There are adjacent nodes i and j at level 1 by the assumption and Lemma 2.

If a node i or j is in *S1a*, it can execute `invite1`. Therefore, both nodes are in *S1b* (Observation 1). Let k be a node pointed by $\texttt{i-ptr}_i$. The level of k is not 5 by Lemma 1.

In case of $\texttt{level}_k = 1$, k is in *S1b* by Observation 1. Let x be a node pointed by $\texttt{i-ptr}_k$. A node k can execute `match2` to make a match with i if $\texttt{level}_x \neq 4$. Therefore, $\texttt{level}_x = 4$ and this implies $\texttt{i-ptr}_x \neq k$ by Lemma 3, and k can execute `cancel1`. In case of $\texttt{level}_k = 2$, k can execute `invite2` if k is in *S2a*. Node i can execute `cancel1` if k is in *S2b* since $\texttt{m-ptr}_k \neq i$ by Lemma 2. If $\texttt{level}_k = 3$ or 4, i can execute `cancel1` since $\texttt{i-ptr}_k \neq i$ by Lemma 3. A contradiction. □

Theorem 2. *A 1-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. 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 from Lemma 2.

Both i and j are at level 2 or higher since they are matching. They are not in *S2a* since they have level 1 neighbors and can execute `invite1` if they are in *S2a*, 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 higher.

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.l.o.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 $\texttt{i-ptr}_k = i$. Therefore, $\texttt{i-ptr}_k = x$ for some $x \neq i$. Since there is no adjacent level 1 nodes by Theorem 1, there is no level 5 node by Lemma 1, 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. □

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) check 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 executes `reset1` or `reset2` in MM1.*

Proof. By contradiction. Assume some nodes execute resets (`reset1` or `reset2`) after executing `match1`, `match2`, `match3`, `migrate1` or `migrate2`. Let i be a node that executes such a move r of a reset first. Let m be the last move of 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 nodes 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. If k causes the reset r, i is at level 4 or 5 at that time. 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 resets while it is at level 1 by Lemma 4 and the validity between i and k is preserved. Node k may move to *S2a* by `migrate1` but never resets 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 Lemma 4. Node i moves to *S2b* after j sets $\texttt{m-ptr}_j = i$ and moves to *S2a* by `match2` or `match3`. 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*, that is i migrates to j, 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 point 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. □

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

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

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

Lemma 7. *Each node execute a progress move at most once in MM1.*

Proof. A progress move changes levels of a node from 1 to 2, and a node never resets if it executes a progress move by Lemma 5. That is the node never goes back to level 1. Therefore, once a node executes a progress move it never executes a progress move again. □

Lemma 8. *In MM1,* `cancel1`*,* `cancel2`*,* `cancel3` *and* `cancel4` *are executed* $O(e)$ *times.*

Proof. In MM1, a node i executes a cancel (`cancel1`, `cancel2`, `cancel3` or `cancel4`) when it is initially possible, some neighbor node executed a cancel, or some neighbor node executed a progress move.

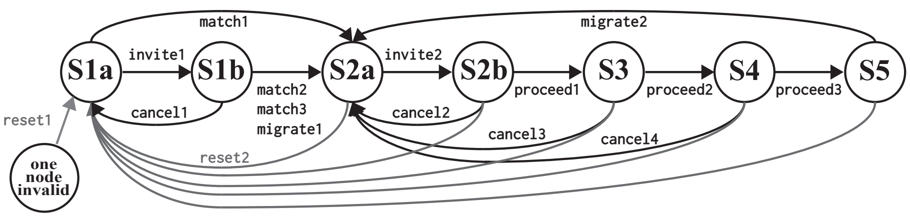

Fig. 5. Transitions of stages

Consider that some node j executes a progress move that changes a stage of j to *S2a*. Nodes that point to j by `i-ptr` will execute a cancel as follows. If such a node k is in *S1b*, k will execute `cancel1`, and if such a node k is in *S2b* or *S3*, k will execute `cancel2` or `cancel3`.

If some node executes `cancel2` or `cancel3`, it causes more cancels. If there is an adjacent node x and trying to increase matches, it will also cancels by `cancel3` or `cancel4`. That cancel may further causes one more cancel. If x already invited some node y to migrate to x, y will execute `cancel1`.

Now we classify cancels with *direct cancels* and *indirect cancels*. The direct cancel is a cancel caused by some progress move or its initial state. The indirect cancel is a cancel caused by a cancel of its neighbor.

From the above observation, any cancel causes at most two indirect cancels. Let deg_j be the degree of j. There are at most deg_j nodes that execute a cancel due to the progress move of j. From Lemma 7, j executes a progress move at most once, and therefore there are at most $\Sigma_{i \in V} deg_i = e$ direct cancels caused by progress moves. Moreover, there are at most n direct cancels caused by initial states. Therefore, the total number of moves by cancels are $O(e)$. □

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

Proof. Let m_1 and m_2 be two consecutive moves by `migrate2` of a node i. The node i moves to *S2a* by m_1 and then invites some neighbor node j at level 1 to migrate to i. Then, node j executes `migrate1` that points to i by `m-ptr`. That is, there is a move by `migrate1` that points to i between two consecutive moves by `migrate2` of node i. Therefore, the total number of moves by `migrate2` $\leq$ the total number of moves by `migrate1` $+n$. From Lemma 7, it is bounded by $O(n)$. □

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. 5 shows 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*, *S4* and *S5* except `reset1`, `reset2`, `cancel1`, `cancel2`, `cancel3`, `cancel4` and `migrate2`. Therefore, if a node does not execute these actions, the number of moves is at most 6.

Let R_i, C_i and M_i be the numbers of moves of a node i by reset (`reset1` or `reset2`), cancel (`cancel1`, `cancel2`, `cancel3` or `cancel4`), and `migrate2`. Let MOV_i denote the total number of moves of a node i. From the observation, it is bounded as follows.

$$MOV_i \leq 7(R_i + C_i + M_i + 1)$$

From Lemmas 6, 8 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 in MM1 can be derived as follows.

$$\Sigma_{i\in V} MOV_i \leq 7(\Sigma_{i\in V} R_i + \Sigma_{i\in V} C_i + \Sigma_{i\in V} M_i + \Sigma_{i\in V} 1) = O(e)$$

Since each node always takes 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. □

5 Conclusion

We proposed a 1-maximal matching algorithm MM1 that is silent and works for any 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 Goddard et al.[6] that is also an anonymous 1-maximal matching algorithm but works for only trees or rings which length is not a multiple of 3 and the time complexity is $O(n^4)$.

References

1. Blair, J.R., Manne, F.: Efficient self-stabilizing algorithms for tree networks. In: Proceedings. 23rd International Conference on Distributed Computing Systems. pp. 20–26. IEEE (2003)
2. Blair, J., Hedetniemi, S., Hedetniemi, S., Jacobs, D.: Self-stabilizing maximum matchings. Congressus Numerantium pp. 151–160 (2001)
3. Chattopadhyay, S., Higham, L., Seyffarth, K.: Dynamic and self-stabilizing distributed matching. In: Proceedings of the twenty-first annual symposium on Principles of distributed computing. pp. 290–297. ACM (2002)
4. Datta, A.K., Larmore, L.L.: Leader election and centers and medians in tree networks. In: Stabilization, Safety, and Security of Distributed Systems, pp. 113–132. Springer (2013)
5. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Commun. ACM 17(11), 643–644 (Nov 1974), http://doi.acm.org/10.1145/361179.361202
6. Goddard, W., Hedetniemi, S.T., Shi, Z., et al.: An anonymous self-stabilizing algorithm for 1-maximal matching in trees. In: Proc. International Conference on Parallel and Distributed Processing Techniques and Applications. pp. 797–803 (2006)

7. Guellati, N., Kheddouci, H.: 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. Hedetniemi, S.T., Jacobs, D.P., Srimani, P.K.: Maximal matching stabilizes in time $O(m)$. Information Processing Letters 80(5), 221–223 (2001)
9. Hsu, S.C., Huang, S.T.: A self-stabilizing algorithm for maximal matching. Information Processing Letters 43(2), 77–81 (1992)
10. Karaata, M.H., Saleh, K.A.: Distributed self-stabilizing algorithm for finding maximum matching. Comput Syst Sci Eng 15(3), 175–180 (2000)
11. Kimoto, M., Tsuchiya, T., Kikuno, T.: 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. Manne, F., Mjelde, M., Pilard, L., Tixeuil, S.: A self-stabilizing 2/3-approximation algorithm for the maximum matching problem. Theoretical Computer Science 412(40), 5515–5526 (2011)
13. Tel, G.: Introduction to distributed algorithms. Cambridge university press (2000)