Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 20, no. 1, pp. 59–78 (2016)
DOI: 10.7155/jgaa.00384

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

Yuma Asada [1] *Fukuhito Ooshita* [1] *Michiko Inoue* [1]

[1]Nara Institute of Science and Technology, Japan

Abstract

We propose a new self-stabilizing 1-maximal matching algorithm which is *silent* and works for any *anonymous* networks without a cycle of length of a multiple of 3 under a central *unfair* daemon. The *1-maximal* matching is a $\frac{2}{3}$-approximation to the maximum matching, and expected to get more matching pairs than a maximal matching, which only guarantees a $\frac{1}{2}$-approximation.

The time complexity of the proposed algorithm is $O(e)$ moves, which is $O(n)$ moves if we restrict the topology to trees or rings whose length is not a multiple of 3, where n and e be the numbers of nodes and edges in a graph, respectively. The best existing result for *1-maximal* matching for anonymous networks is an algorithm of Goddard et al. [8] which works for anonymous trees and anonymous rings whose length is not a multiple of 3 under a central daemon, and the time complexity is $O(n^4)$ moves. Therefore, the result in this paper is a significant improvement from the best existing results.

Submitted:	Reviewed:	Revised:	Accepted:	Final:
March 2015	August 2015	August 2015	September 2015	January 2016
		Published:		
		February 2016		

Article type: Regular paper
Communicated by: M. S. Rahman and E. Tomita

This work was supported by JSPS KAKENHI Grant Numbers 26330084 and 15H00816.
E-mail addresses: asada.yuma.ar4@is.naist.jp (Yuma Asada) f-oosita@is.naist.jp (Fukuhito Ooshita) kounoe@is.naist.jp (Michiko Inoue)

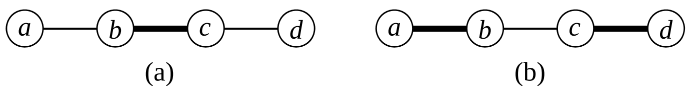

Figure 1: A maximal matching and a 1-maximal matching. The matching in (a) is maximal but not 1-maximal. The matching in (b) is 1-maximal.

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 from any inconsistent system configuration and stabilize to consistent system configuration without restarting program execution.

Maximal or *maximum 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. A matching is maximal if no proper superset of it is a matching as well, and it is maximum if its cardinality is the largest among all matchings. A matching can be used in distributed applications where pairs of nodes are required. For example, when each server gives some service to one client, pairs of a server and a client can be constructed by a matching. Another application is communication scheduling in wireless networks. Since a matching represents pairs of a sender and a receiver that can communicate at the same time, many communication scheduling algorithms utilize a matching.

This paper proposes an efficient anonymous self-stabilizing algorithm for *1-maximal matching*. A 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 $M - \{e\}$ (See Fig. 1). A 1-maximal matching is a $\frac{2}{3}$-approximation to the maximum matching[1], and expected to find more matching pairs than a *maximal matching*, which only guarantees a $\frac{1}{2}$-approximation to the maximum matching.

Related Works Self-stabilizing algorithms for the maximum and maximal matching problems have been well studied [9]. Table 1 summarizes the results, where n and e denote the numbers of nodes and edges, respectively.

For maximum, maximal, and 1-maimal matching problems, some algorithm have been proposed for various assumptions. One important assumption is that the network is *anonymous* or not. In anonymous networks an algorithm cannot use global IDs of nodes, while it can use them in non-anonymous networks. Another important assumption is a daemon [6], which decides an execution of an algorithm. Most algorithms assume a *central daemon* or a *distributed daemon*. A central daemon chooses one node to move at the same time, while a distributed daemon can choose one or more nodes at the same time. Note that, if each node changes its state depending on only itself and its neighboring nodes,

[1]We say matching M is an α-approximation to the maximum matching if $|M| \geq \alpha|M_{max}|$ holds where M_{max} is a maximum matching.

each execution under a distributed daemon is equivalent to an execution under a central daemon unless neighboring nodes are chosen at the same time. Therefore, most algorithms (including our proposed algorithm) for a central daemon still work under a distributed daemon with such a restriction. Some algorithms assume a *read/write* daemon, which is a generalized one of a distributed daemon. The read/write daemon can execute read and write operations of each node separately.

The time complexity is measured in *moves* or *rounds*. A move is the execution of one action by one node, while a round is a minimal sub-sequence of an execution in which every node has at least one chance to execute some action. That is, when a node can execute an action at the beginning of a round, it takes at least one action or becomes ineligible to execute it by actions of neighboring nodes during the round.

Blair and Manne [2] 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 a bottom-up algorithm to construct a rooted tree, and showed that the maximum matching algorithm by Blair et al. [1] can be combined with the proposed bottom-up algorithm without increasing the time complexity. For anonymous tree networks, Karaata et al. [12] proposed a maximum matching algorithm with $O(n^4)$ moves under a central daemon. For anonymous bipartite networks, Chattopadhyay et al. [3] proposed a maximum matching algorithm with $O(n^2)$ rounds under a central daemon on the assumption that each node knows its bipartition, that is, which bipartite group the node belongs to.

Hsu and Huang [11] 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 [17] and Kimoto et al. [13] and $O(e)$ moves by Hedetniemi et al. [10].

Fredrik Manne et al. [14] also proposed a *maximal* matching algorithm. It works for non-anonymous networks with arbitrary topology under a distributed daemon. They showed the time complexity of $O(e)$ moves. The time complexity is the same as one of Hsu and Huang, however, they are different in network's

Table 1: Self-stabilizing matching algorithms.

Reference	Matching	Topology	Anonymity	Daemon	Complexity
[2]	maximum	tree	no	read/write	$O(n^2)$ moves
[12]	maximum	tree	yes	central	$O(n^4)$ moves
[3]	maximum	bipartite*	yes	central	$O(n^2)$ rounds
[11]	maximal	arbitrary	yes	central	$O(e)$ moves
[14]	maximal	arbitrary	no	distributed	$O(e)$ moves
[8]	1-maximal	tree, ring**	yes	central	$O(n^4)$ moves
[15]	1-maximal	arbitrary	no	distributed	$O(2^n \cdot \Delta n)$ moves
this paper	1-maximal	arbitrary**	yes	central	$O(e)$ moves

* Each node knows its bipartition.

** A network does not contain a cycle of length of a multiple of 3.

anonymity and type of scheduler.

Self-stabilizing *maximal* matching algorithms with additional properties were also proposed in literature. Devismes et al. [4] proposed a communication-efficient maximal matching algorithm for colored networks with arbitrary topology under a distributed daemon. Note that colored networks mean every node has a color such that neighboring nodes have different colors. The time complexity is $O(\Delta \cdot n)$ rounds, where Δ is the maximum degree of nodes. The important property of this algorithm is communication efficiency after stabilization. In this algorithm a fraction of the nodes communicates with exactly one neighboring node after stabilization while in most algorithms every node communicates with every neighboring node. Dubois et al. [7] proposed a Byzantine-tolerant maximal matching algorithm for anonymous networks with arbitrary topology under a central daemon. This algorithm guarantees that, even if Byzantine-faulty nodes continue to change their states arbitrarily, correct nodes sufficiently distant from Byzantine-faulty nodes construct a maximal matching in finite moves.

Goddard et al. [8] 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. Note that, since a maximum matching is also 1-maximal, it is impossible to construct a maximum matching for such rings. This is the reason why the algorithm for anonymous bipartite networks, which include rings with length of a multiple of 3, requires each node to know additional information such as its bipartition. Manne et al. [15] 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.

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 length of a multiple of 3 under a central *unfair* daemon. We show that the time complexity of the proposed algorithm is $O(e)$ moves.

The algorithm of Goddard et al. [8] works for anonymous trees and anonymous rings whose length is not a multiple of 3 under a central daemon, and the time complexity is $O(n^4)$ moves. Our proposed algorithm has time complexity of $O(n)$ moves if the topology is restricted to trees or rings whose length is not a multiple of 3. That is, we significantly improve the existing best 1-maximal matching algorithm for anonymous networks.

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 proofs for its correctness and performance are given in Section 4. Finally Section 6 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 state by actions. Each node has a set of actions, and a collection of actions of nodes is called a *distributed algorithm*.

In this paper, we consider *state-reading model* as a communication model where each node can directly read the state of its neighboring nodes. 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 neighboring nodes, 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 recognize whether its neighboring node points to itself, some other nodes, or no node[2].

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.

[2]This assumption is identical to previous works for anonymous networks. One way to implement this assumption is to execute an edge-coloring algorithm [16], that is, to assign a color to each edge so that no two edges with the same color share a node. In this case, since $O(\Delta)$ colors are sufficient, each pointer can be implemented by $O(\log \Delta)$ bits. Although an edge-coloring algorithm incurs some extra steps, we ignore them in the complexity analysis as in previous works.

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 neighboring node, and, *S2b*, *S3*, *S4*, and *S5* mean the node is trying to increase matches. A node i has three variables; $\texttt{level}_i$, $\texttt{m-ptr}_i$, $\texttt{i-ptr}_i$, and its stage is determined by values of these three variables. 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 neighboring node. Fig.2 shows how free nodes make a match. Consider two nodes i and j where their levels are 1 and neither of them point to any node (Fig.2(a)). When the free node i finds the free neighboring node j, i invites j by $\texttt{i-ptr}_i$ in Fig.2(b) (i is in *S1b*). Then the invited node j updates its level to 2 and points to i by $\texttt{m-ptr}_j$ to accept the invitation in Fig.2(c) (j is in *S2a*). Finally i points to j by $\texttt{m-ptr}_i$ to make a match in Fig.2(d) (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 $(i, j) \in M$. There are corresponding four actions as follows. The pseudo code of MM1 is shown in Fig.4 and Fig.5.

- `invite1`: A free node i invites a free neighboring node x by pointing to x with `i-ptr`. The guard of `invite1` includes conditions *S1a(i)* and *no_invalid1_neighbor(i)*. They are related to reset actions and explained later.
- `match1`: If a free node i is invited by some neighboring free node x, i points to x with `m-ptr`.
- `match2`: In the case where a free node i is invited by some neighboring free node x and i is also inviting a neighboring node k, if k's level is 1 or 3, i points to x with `m-ptr`. The constraint on k's level will be explained later.
- `match3`: If a free node i is inviting a neighboring node k and k is pointing to i with `m-ptr`, i points to k with `m-ptr` to make a match with k.

S2b, S3, S4, S5 Matching nodes try to increase the number of matches if they have free neighboring nodes. Fig.3 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.3(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 i and j notice that the other node invite free neighboring

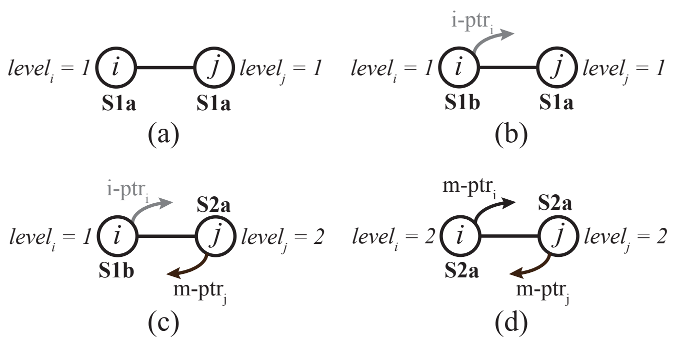

Figure 2: Making a match between free nodes

nodes, they change their level to 3 (i and j are in *S3*). That indicates they are ready to be approved as in Fig.3(b). Then k and l point to the inviting nodes by `i-ptr` to approve their invitations (k and l are in *S1b*). Nodes i and j change their level to 4 if the neighbors approve the invitations (i and j are in *S4*) as in Fig.3(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.3(d). Then they create new matches with the free nodes, where k and l first move to *S2a* (Fig.3(e)) and then i and j move to *S2a* (Fig.3(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*. There are corresponding 7 actions as follows.

- `invite2`: If a matching node i has a free neighboring node x, i invites x to increase matches by migrating to x.
- `proceed1`: If a matching node i is inviting some neighboring node and its matching partner j is also inviting its neighboring node, i becomes level 3 to indicate that i's invitation is ready to be approved.
- `approve1`: If a free node i is invited by some non-free (matching) node x and x is at level 3, i approves the invitation from x by pointing x with `i-ptr`.
- `proceed2`: If a matching node i is inviting a free neighboring node k and k has approved the invitation from i, i becomes level 4.
- `proceed3`: If a matching node i is at level 4 and its matching partner j is level 4 or 5, i becomes level 5 to indicate that it is ready to break a match between i and j.
- `migrate1`: If a free node i is invited by some non-free node k and k is at level 5, i points to k with `m-ptr`.

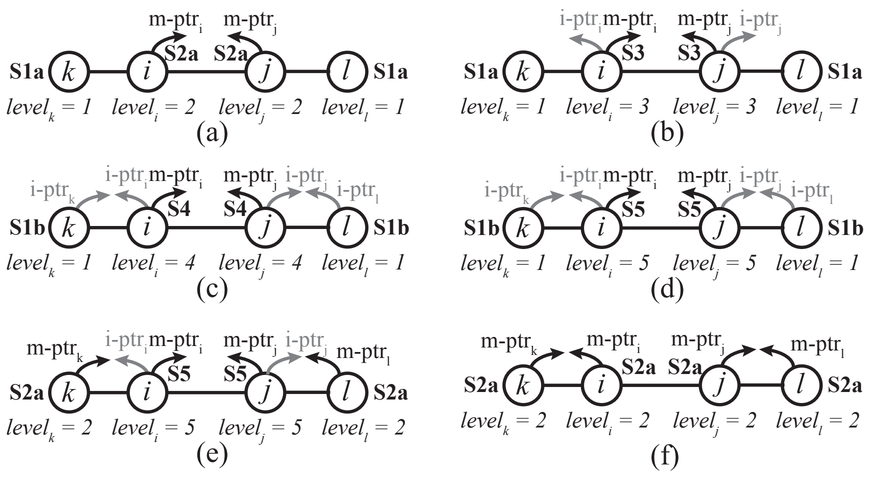
m-ptr$_i$
m-ptr$_j$
S1a
S2a
S2a
S1a
$level_k = 1$ $level_i = 2$ $level_j = 2$ $level_l = 1$
(a)
i-ptr$_i$ m-ptr$_i$
m-ptr$_j$ i-ptr$_j$
S1a
S3
S3
S1a
$level_k = 1$ $level_i = 3$ $level_j = 3$ $level_l = 1$
(b)
i-ptr$_k$
i-ptr$_i$ m-ptr$_i$
m-ptr$_j$ i-ptr$_j$
i-ptr$_l$
S1b
S4
S4
S1b
$level_k = 1$ $level_i = 4$ $level_j = 4$ $level_l = 1$
(c)
S1b
S5
S5
S1b
$level_k = 1$ $level_i = 5$ $level_j = 5$ $level_l = 1$
(d)
m-ptr$_k$
m-ptr$_l$
S2a
S5
S5
S2a
$level_k = 2$ $level_i = 5$ $level_j = 5$ $level_l = 2$
(e)
S2a
S2a
S2a
S2a
$level_k = 2$ $level_i = 2$ $level_j = 2$ $level_l = 2$
(f)

Figure 3: Increasing matches

- `migrate2`: If a matching node i is pointed by a neighboring node k with `m-ptr` and its matching node j is at level 5 or has already migrated, i migrates to k by pointing to k with `m-ptr` to make a new match.

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 of a node means that the combination of its three variables 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 by `m-ptr` should point to i by `m-ptr` at level 2, 3, or 4, or by `i-ptr` at level 1 or 5. The full definition of the validity function is shown in Fig.4. A node does not move while some neighboring node is one node invalid. There are corresponding two actions as follows.

- `reset1`: If a node i finds it does not satisfy one node validity, i resets itself that means i moves to a stage *S1a*.
- `reset2`: If a node i finds it does not satisfy two node validity, i resets itself.

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. There are corresponding 4 actions as follows.

- `cancel1`: A node i in a stage *S1b* is inviting a free node to make a match between free nodes or inviting (approving) a non-free node that is also inviting i to migrate to i. If the invited free node k has made a match with another node and become level 2, i cancels its invitation and goes back to *S1a* . If the invited non-free node k has canceled the migration to i, k is at level 2 and has a matching partner, or at level 3 or higher but not inviting i any longer. In this case, i also cancels its invitation.

- `cancel2`: A node i in a stage *S2b* is inviting some free node k to migrate it. If k has become level 2 or higher, it means that k is matching with another node. In this case, i cancels its invitation and goes back to *S2a*.
- `cancel3`: A node i becomes level 3 if it notices that both i and its matching partner are inviting some free nodes to migrate to them. If its matching partner has canceled the invitation and gone back to *S2a*, i also cancels the invitation and goes back to *S2a*.
- `cancel4`: A node i becomes level 4 after level 3. Similarly to `cancel3`, if its matching partner has canceled the invitation, i also cancels the invitation and goes back to *S2a*.

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.4, and a code of MM1 is shown in Fig.5. 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 neighboring node points to itself, some other node, or no node by pointers `i-ptr` and `m-ptr`.

4 Correctness

First, we show that 1-maximal matching is constructed once the network reaches to a terminal configuration where the topology is required to exclude any cycle of length of a multiple of 3.

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 $\texttt{level}_k = 2 \wedge \texttt{m-ptr}_k = i$ 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 neighboring node by `m-ptr` is 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),

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): $\texttt{level}_i = 2 \wedge \texttt{m-ptr}_i = j \wedge \texttt{i-ptr}_i =\perp$
S2b_valid(i,j,k): $\texttt{level}_i = 2 \wedge \texttt{m-ptr}_i = j \wedge \texttt{i-ptr}_i = k \wedge j \neq k$
S3_valid(i,j,k): $\texttt{level}_i = 3 \wedge \texttt{m-ptr}_i = j \wedge \texttt{i-ptr}_i = k \wedge j \neq k$
S4_valid(i,j,k): $\texttt{level}_i = 4 \wedge \texttt{m-ptr}_i = j \wedge \texttt{i-ptr}_i = k \wedge j \neq k$
S5_valid(i,j,k): $\texttt{level}_i = 5 \wedge \texttt{m-ptr}_i = j \wedge \texttt{i-ptr}_i = k \wedge j \neq k$

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): $\exists j \in N(i)$ *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): $\exists j,k \in N(i)$ *S5_valid(i,j,k)*
valid1(i): *S1a_valid1(i)* $\vee$ *S1b_valid1(i)* $\vee$ *S2a_valid1(i)* $\vee$ *S2b_valid1(i)* $\vee$ *S3_valid1(i)* $\vee$ *S4_valid1(i)* $\vee$ *S5_valid1(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)* $\wedge ((\texttt{level}_j = 2 \wedge \texttt{m-ptr}_j = i) \vee (\texttt{level}_j = 3 \wedge \texttt{m-ptr}_j = i) \vee (\texttt{level}_j = 4 \wedge \texttt{m-ptr}_j = i) \vee (\texttt{level}_j = 1 \wedge \texttt{i-ptr}_j = i) \vee (\texttt{level}_j = 5 \wedge \texttt{i-ptr}_j = i)))$
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)* $\wedge \texttt{level}_j \geq 2 \wedge \texttt{m-ptr}_j = i \wedge \texttt{level}_k = 1 \wedge \texttt{i-ptr}_k = i)$
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)* $\vee$ *S1b(i)* $\vee$ *S2a(i)* $\vee$ *S2b(i)* $\vee$ *S3(i)* $\vee$ *S4(i)* $\vee$ *S5(i)*
invalid(i): $\neg$ *valid(i)*

Statement Macros
make_match: $\texttt{i-ptr}_i =\perp, \texttt{m-ptr}_i = k, \texttt{level}_i = 2$
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)*

Figure 4: Variables, validity functions, statement macros and guard function

Reset
$\texttt{reset1}$ :: $\textit{invalid1(i)} \mapsto \texttt{reset\_state}$
$\texttt{reset2}$ :: $\textit{invalid(i)} \wedge \textit{no\_invalid1\_neighbor(i)} \mapsto \texttt{reset\_state}$

S1a
$\texttt{match1}$ :: $\textit{S1a(i)} \wedge \textit{no\_invalid1\_neighbor(i)} \wedge \exists x \in N(i)(\texttt{i-ptr}_x = i \wedge \texttt{level}_x = 1) \mapsto \texttt{i-ptr}_i = \perp, \texttt{m-ptr}_i = x, \texttt{level}_i = 2$
$\texttt{approve1}$ :: $\textit{S1a(i)} \wedge \textit{no\_invalid1\_neighbor(i)} \wedge \exists x \in N(i)(\texttt{i-ptr}_x = i \wedge \texttt{level}_x = 3) \mapsto \texttt{i-ptr}_i = x$
$\texttt{invite1}$ :: $\textit{S1a(i)} \wedge \textit{no\_invalid1\_neighbor(i)} \wedge \exists x \in N(i)\texttt{level}_x = 1 \mapsto \texttt{i-ptr}_i = x$

S1b
$\texttt{match2}$ :: $\textit{S1b(i)} \wedge \textit{no\_invalid1\_neighbor(i)} \wedge \exists x \in N(i)(\texttt{i-ptr}_x = i \wedge \texttt{level}_x = 1) \wedge \exists k \in N(i)(\textit{S1b\_valid(i,k)} \wedge (\texttt{level}_k = 1 \vee \texttt{level}_k = 3)) \mapsto \texttt{i-ptr}_i = \perp, \texttt{m-ptr}_i = x, \texttt{level}_i = 2$
$\texttt{match3}$ :: $\textit{S1b(i)} \wedge \textit{no\_invalid1\_neighbor(i)} \wedge \exists k \in N(i)(\textit{S1b\_valid(i,k)} \wedge \texttt{m-ptr}_k = i \wedge \texttt{level}_k = 2) \mapsto \texttt{make\_match}$
$\texttt{migrate1}$ :: $\textit{S1b(i)} \wedge \textit{no\_invalid1\_neighbor(i)} \wedge \exists k \in N(i)(\textit{S1b\_valid(i,k)} \wedge \texttt{i-ptr}_k = i \wedge \texttt{level}_k = 5) \mapsto \texttt{make\_match}$
$\texttt{cancel1}$ :: $\textit{S1b(i)} \wedge \textit{no\_invalid1\_neighbor(i)} \wedge \exists k \in N(i)(\textit{S1b\_valid(i,k)} \wedge ((\texttt{level}_k = 2 \wedge \texttt{m-ptr}_k \neq i) \vee (\texttt{level}_k = 3 \wedge \texttt{i-ptr}_k \neq i) \vee (\texttt{level}_k = 4 \wedge \texttt{i-ptr}_k \neq i) \vee (\texttt{level}_k = 5 \wedge \texttt{i-ptr}_k \neq i))) \mapsto \texttt{i-ptr}_i = \perp$

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

S2b
$\texttt{cancel2}$ :: $\textit{S2b(i)} \wedge \textit{no\_invalid1\_neighbor(i)} \wedge \exists j,k \in N(i)(\textit{S2b\_valid(i,j,k)} \wedge \texttt{level}_k \geq 2) \mapsto \texttt{abort\_exchange}$
$\texttt{proceed1}$ :: $\textit{S2b(i)} \wedge \textit{no\_invalid1\_neighbor(i)} \wedge \exists j,k \in N(i)(\textit{S2b\_valid(i,j,k)} \wedge \texttt{i-ptr}_j \neq \perp) \mapsto \texttt{level}_i = 3$

S3
$\texttt{cancel3}$ :: $\textit{S3(i)} \wedge \textit{no\_invalid1\_neighbor(i)} \wedge \exists j,k \in N(i)(\textit{S3\_valid(i,j,k)} \wedge ((\texttt{level}_j = 2 \wedge \texttt{i-ptr}_j = \perp) \vee \texttt{level}_k \geq 2)) \mapsto \texttt{abort\_exchange}$
$\texttt{proceed2}$ :: $\textit{S3(i)} \wedge \textit{no\_invalid1\_neighbor(i)} \wedge \exists j,k \in N(i)(\textit{S3\_valid(i,j,k)} \wedge \texttt{i-ptr}_k = i \wedge \texttt{level}_k = 1) \mapsto \texttt{level}_i = 4$

S4
$\texttt{cancel4}$ :: $\textit{S4(i)} \wedge \textit{no\_invalid1\_neighbor(i)} \wedge \exists j,k \in N(i)(\textit{S4\_valid(i,j,k)} \wedge \texttt{level}_j = 2 \wedge \texttt{i-ptr}_j = \perp) \mapsto \texttt{abort\_exchange}$
$\texttt{proceed3}$ :: $\textit{S4(i)} \wedge \textit{no\_invalid1\_neighbor(i)} \wedge \exists j,k \in N(i)(\textit{S4\_valid(i,j,k)} \wedge (\texttt{level}_j = 4 \vee \texttt{level}_j = 5)) \mapsto \texttt{level}_i = 5$

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

Figure 5: Algorithm MM1

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 `invite2` 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 `approve1` if $\texttt{i-ptr}_k = \perp$, 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 node 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, both i and j are not in *S3*.

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

Now we show that MM1 always brings the network to a terminal configuration and evaluate the time complexity. In this part, the topology does not need to be restricted. That is, MM1 always brings the network to a terminal configuration with $O(e)$ moves even if the network include a cycle of length of a multiple of 3.

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 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. 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 between 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 executes 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 neighboring node executed a cancel, or some neighboring node executed a progress move.

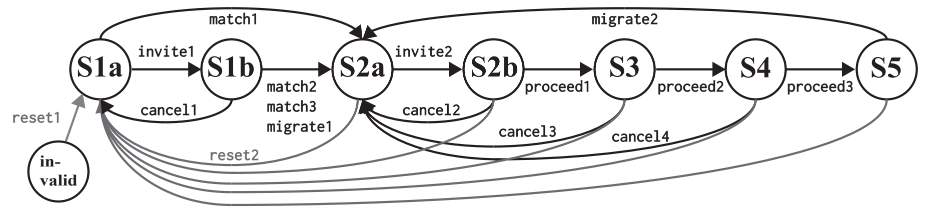

Figure 6: Transitions of stages

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 trying to increase matches, it will also cancel by `cancel3` or `cancel4`. That cancel may further cause 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 neighboring 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 $O(n)$. □

Lemma 10 *MM1 is silent and takes $O(e)$ moves to reach a terminal configuration.*

Proof: Fig. 6 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 and this also implies MM1 is silent. □

Now the final theorem is derived from Theorem 2 and Lemma 10 .

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: Lemma 10 denotes that the system eventually reaches to a terminal configuration. In addition, Theorem 2 denotes a 1-maximal matching is constructed in any terminal configuration for any graphs without a cycle of length of a multiple of 3. Hence, MM1 constructs a 1-maximal matching in $O(e)$ moves for any graphs without a cycle of length of a multiple of 3. □

The space complexity of MM1 depends on the implementation of pointers. When each pointer is implemented with a $O(\log \Delta)$-bit memory, the space complexity per node is $O(\log \Delta)$ because each node requires one 3-bit variable (`level`) and two pointers (`m-ptr` and `i-ptr`).

5 Observation

It is shown that there is no self-stabilizing 1-maximal matching algorithm for anonymous ring networks whose length of a multiple of 3 [8]. Therefore, both our algorithm and the algorithm of Goddard et al. [8] exclude the topology including a cycle of length of a multiple of 3. In this Section, we observe what happen in MM1 and the algorithm of Goddard et al. [8] if the network includes a cycle of length of a multiple of 3.

We mentioned that proposed algorithm MM1 is silent for any network in Lemma 10. That is, MM1 always brings the network to a terminal configuration even if it includes a cycle of length of a multiple of 3.

However, we could not always obtain 1-maximal matching unfortunately. Fig.7 shows an example terminal configuration that could not construct either 1-maximal or maximal matching. In this example, bold lines between nodes

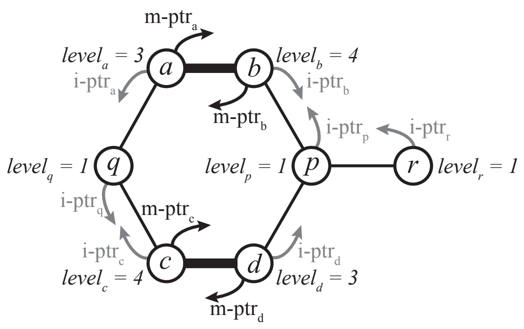

Figure 7: A terminal configuration of MM1

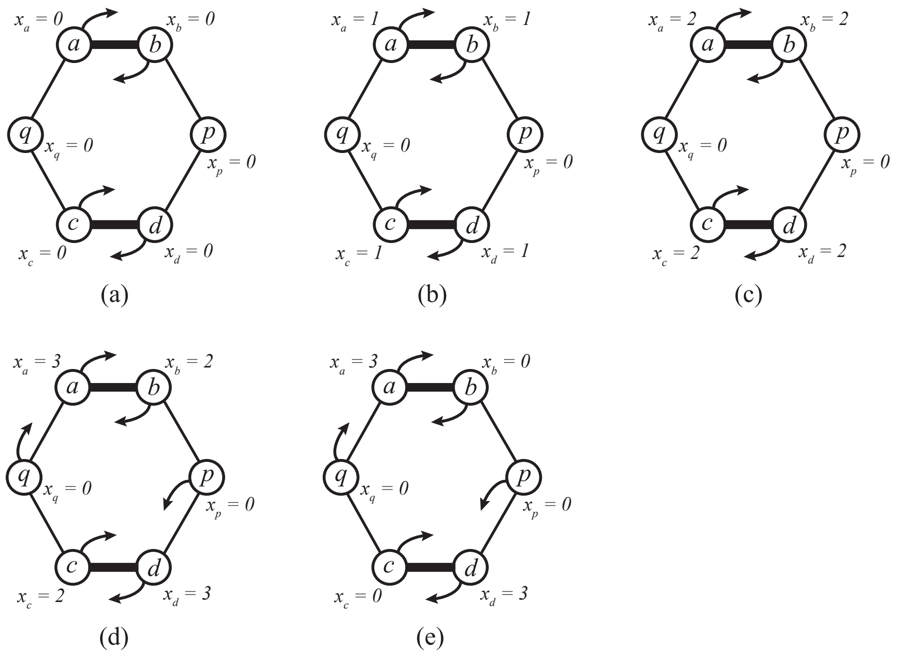
(a) x_a = 0 x_b = 0 x_q = 0 x_p = 0 x_c = 0 x_d = 0
(b) x_a = 1 x_b = 1 x_q = 0 x_p = 0 x_c = 1 x_d = 1
(c) x_a = 2 x_b = 2 x_q = 0 x_p = 0 x_c = 2 x_d = 2
(d) x_a = 3 x_b = 2 x_q = 0 x_p = 0 x_c = 2 x_d = 3
(e) x_a = 3 x_b = 0 x_q = 0 x_p = 0 x_c = 0 x_d = 3

Figure 8: An example of an infinitely long execution in the algorithm of Goddard et al.

represent edges of a matching. In this configuration, nodes p and q are waiting for nodes b and c to proceed to *S5* or cancel to increase matches, respectively. Nodes b and c are waiting for node a and d to proceed to *S4* or cancel to increase matches, respectively. Nodes a and d are waiting for nodes p and q to approve their invitations or for b and c to cancel to increase matches, respectively. In addition, a node r is waiting for a node p to accept its invitation. Therefore, no node can move, that is, the network is in a terminal configuration.

On the other hand, the algorithm of Goddard et al. [8] is not silent for some networks. That is, nodes can move infinitely often in some situation and could not reach to any terminal configuration if the network has a cycle of a multiple of 3.

In the algorithm of Goddard et al., each node i has one pointer and variable $x_i \in \{0, 1, 2, 3\}$. The pointer is used as `m-ptr` and the variable x is used as *level* in MM1. Therefore, nodes can easily change partners and easily try and cancel to increase matches. In their algorithm, if two nodes point to each other and difference of values of x is within 1, they are recognized to be matching. Matching nodes can increase matches when both nodes have free neighbors. In this case, the two nodes synchronously increase values of x, and once both x values reach to 3, they can migrate to their free neighbors.

Fig. 8 shows an example of an infinitely long execution. It starts from an initial configuration showed in Fig. 8 (a). In the configuration, nodes a and b, c and d are matching, respectively, and nodes p and q are free. Therefore, four nodes a, b, c and d try to increase matches. Matching nodes increment a value of x to 1 if they have free neighbors as in Fig. 8 (b). Then, these node can further increment a value of x to 2 if both two matching nodes have incremented values of x to at least 1 as in Fig. 8 (c). They can further increment a value of x to 3 if their free neighbors point to them. In Fig. 8 (d), only nodes a and d have incremented values of x to 3, while b and c could not increment the values since nodes p and q do not point to them. In this situation, nodes b and c cancel to increase matches and reset values of x to 0 as in Fig. 8 (e). Then nodes a and d subsequently cancel to increase matches and then nodes q and p further follow them. Finally, the configuration goes back to a configuration in Fig. 8 (a), and the above scenario can be infinitely repeated.

The observation implies that the algorithm of Goddard et al. [8] is not silent. This observation gives us a question that which algorithm is preferred for the network that includes a cycle with length of a multiple of 3. There is a tradeoff as follows. Our proposed algorithm MM1 preserves a property of silence, and therefore, it can save communication after the network is stabilized. While, the algorithm Goddard et al. [7] can move infinitely often in some situation. Therefore, it cannot save communication but it might be valuable information for an upper layer algorithm to know that the stabilization has not succeeded yet.

6 Conclusion

We proposed a 1-maximal matching algorithm MM1 that is silent and works for any anonymous networks without a cycle of 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. [8] that is the existing best anonymous 1-maximal matching algorithm but works for only trees or rings whose length is not a multiple of 3 and the step complexity is $O(n^4)$.

We also observed what happen for networks with a cycle of length of a multiple of 3. In the proposed MM1, though it cannot always reach to 1-maximal matching, we found it is still silent for general networks. On the other hand, we showed that the algorithm of Goddard et al. [8] is not silent if the network includes a cycle of length of a multiple of 3.

The proposed MM1 could not always achieve *maximum matching* but it is superior to existing maximum matching algorithms [2, 3, 12] with respect to the step complexity. Regarding topology, though it is proved that there is no anonymous self-stabilizing 1-maximal matching algorithm for cycles with length of a multiple of 3 [8], if we give up the anonymity, there exists a 1-maximal matching algorithm for general network [15]. However, it has an exponential step complexity. That is there remains a big gap between networks without a cycle of length of a multiple of 3 and general networks.

The future work includes to propose a self-stabilizing 1-maximal matching algorithm for anonymous networks that works under a distributed daemon.

References

[1] J. Blair, S. Hedetniemi, S. Hedetniemi, and D. Jacobs. Self-stabilizing maximum matchings. *Congressus Numerantium*, pages 151–160, 2001.

[2] J. R. Blair and F. Manne. Efficient self-stabilizing algorithms for tree networks. In *Proceedings of 23rd International Conference on Distributed Computing Systems*, pages 20–26. IEEE, 2003. doi:10.1109/ICDCS.2003.1203448.

[3] S. Chattopadhyay, L. Higham, and K. 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. doi:10.1145/571825.571877.

[4] S. Devismes, T. Masuzawa, and S. Tixeuil. Communication efficiency in self-stabilizing silent protocols. In *Proceedings of 23rd International Conference on Distributed Computing Systems*, pages 474–481. IEEE, 2009. doi:10.1109/ICDCS.2009.24.

[5] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974. doi:10.1145/361179.361202.

[6] S. Dubois and S. Tixeuil. A taxonomy of daemons in self-stabilization. *CoRR*, abs/1110.0334, 2011. URL: http://arxiv.org/abs/1110.0334.

[7] S. Dubois, S. Tixeuil, and N. Zhu. The byzantine brides problem. In *Fun with Algorithms*, pages 107–118. Springer, 2012. doi:10.1007/978-3-642-30347-0_13.

[8] W. Goddard, S. T. Hedetniemi, Z. Shi, et al. An anonymous self-stabilizing algorithm for 1-maximal matching in trees. In *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 797–803, 2006.

[9] N. Guellati and H. 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. doi:10.1016/j.jpdc.2009.11.006.

[10] S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Maximal matching stabilizes in time $O(m)$. *Information Processing Letters*, 80(5):221–223, 2001. doi:10.1016/S0020-0190(01)00171-5.

[11] S.-C. Hsu and S.-T. Huang. A self-stabilizing algorithm for maximal matching. *Information Processing Letters*, 43(2):77–81, 1992. doi:10.1016/0020-0190(92)90015-N.

[12] M. H. Karaata and K. A. Saleh. Distributed self-stabilizing algorithm for finding maximum matching. *Computer Systems Science and Engineering*, 15(3):175–180, 2000.

[13] M. Kimoto, T. Tsuchiya, and T. Kikuno. The time complexity of Hsu and Huang's self-stabilizing maximal matching algorithm. *IEICE Transactions on Infrmation and Systems*, E93-D(10):2850–2853, 2010. `doi:10.1587/transinf.E93.D.2850`.

[14] F. Manne, M. Mjelde, L. Pilard, and S. Tixeuil. A new self-stabilizing maximal matching algorithm. *Theoretical Computer Science*, 410(14):1336–1345, 2009. `doi:10.1016/j.tcs.2008.12.022`.

[15] F. Manne, M. Mjelde, L. Pilard, and S. Tixeuil. A self-stabilizing 2/3-approximation algorithm for the maximum matching problem. *Theoretical Computer Science*, 412(40):5515–5526, 2011. `doi:10.1016/j.tcs.2011.05.019`.

[16] T. Masuzawa and S. Tixeuil. A self-stabilizing link-coloring protocol resilient to unbounded byzantine faults in arbitrary networks. In *Principles of Distributed Systems*, pages 283–298. Springer, 2005. `doi:10.1007/11795490_11`.

[17] G. Tel. *Introduction to distributed algorithms.* Cambridge university press, 2000.