

# A Robust Distributed Generalized Matching Protocol that Stabilizes in Linear Time\*

Wayne Goddard, Stephen T. Hedetniemi, David P. Jacobs and Pradip K. Srimani

Department of Computer Science

Clemson University

Clemson, SC 29634-0974

{goddard,hedet,dpj,srimani}@cs.clemson.edu

## Abstract

We present a self-stabilizing algorithm for finding a generalized maximal matching ( $b$ -matching) in an arbitrary distributed network. We show that the algorithm converges in  $O(m)$  moves under an unfair central demon independent of the  $b$ -values at different nodes. The algorithm is capable of working with multiple types of demons (schedulers) as is the most recent algorithm in [1, 2].

## 1 Introduction

Most of the essential fundamental services for networked distributed systems (mobile or wired) involve maintaining a global predicate over the entire network (defined by some invariance relation on the global state of the network) by using local knowledge at each of the participating nodes. For example, a minimal spanning tree must be maintained to minimize latency and bandwidth requirements of multicast/broadcast messages or to implement echo-based distributed algorithms [3]; a minimal dominating set must be maintained to optimize the number and the locations of the resource centers in a network [4]; an  $(r, d)$  configuration must be maintained in a network where various resources must be allocated but all nodes have a fixed capacity  $r$  [5]; a minimal coloring of the nodes must be maintained [6].

The traditional approach to building fault-tolerant, distributed systems uses *fault masking*. It is *pessimistic* in the sense that it assumes a worst case scenario and protects the system against such an eventuality. Validity is guaranteed in the presence of faulty processes, which necessitates restrictions on the number of faults and on the fault model. But fault masking is not free; it requires additional hardware or software, and it considerably increases the cost of

the system. This additional cost may not be an economic option, especially when most faults are transient in nature and a temporary unavailability of a system service is acceptable for a short period of time. The paradigm of *Self-stabilization* can cope with the transient faults in a very elegant and cost effective way to design fault tolerant protocols for networked computer systems. Self-stabilization is an *optimistic* way of looking at system fault tolerance, because it provides a built-in safeguard against transient failures that might corrupt the data in a distributed system. Although the concept was introduced by Dijkstra in 1974 [7], and Lamport [8] showed its relevance to fault tolerance in distributed systems in 1983, serious work only began in the late nineteen-eighties. A good survey of self-stabilizing algorithms can be found in [9]. Herman's bibliography [10] also provides a fairly comprehensive listing of most papers in this field. Because of the size and nature of many ad hoc and geographically distributed systems, communication links are unreliable. The system must therefore be able to adjust when faults occur. But 100% fault tolerance is not warranted. The promise of self-stabilization, as opposed to fault masking, is to recover from failure in a reasonable amount of time and without intervention by any external agency. Since the faults are transient (eventual repair is assumed), it is no longer necessary to assume a bound on the number of failures.

A fundamental idea of self-stabilizing algorithms is that the distributed system may be started from an arbitrary global state. After a finite amount of time the system reaches a correct global state, called a *legitimate* or *stable* state. An algorithm is self-stabilizing if (i) for any initial illegitimate state it reaches a legitimate state after a finite number of node moves, and (ii) for any legitimate state and for any move allowed by that state, the next state is a legitimate state. A self-stabilizing system does not guarantee that the system is able to operate properly when a node continuously injects faults in the system (Byzantine fault)

\*This work has been supported by NSF grant # ANI-0073409 and NSF grant # ANI-0218495

or when communication errors occur so frequently that the new legitimate state cannot be reached. While the system services are unavailable when the self-stabilizing system is in an illegitimate state, the repair of a self-stabilizing system is simple; once the offending equipment is removed or repaired the system provides its service after a reasonable time.

In this paper we propose a new simple and elegant self-stabilizing algorithm to solve the generalized matching problem in an arbitrary graph. Given an undirected graph  $G = (V, E)$ , a **matching** is defined to be a subset  $M$  of edges  $(i, j) \in E$ , where  $i, j \in V$  such that for all nodes  $i \in V$  at most one edge of  $M$  is incident on  $i$ . A matching  $M$  is maximal if there does not exist another matching  $M'$  such that  $M' \supset M$ . Authors in [11] have presented a self-stabilizing algorithm for finding a *maximal matching* in a distributed network  $G = (V, E)$ ; complexity analysis of the algorithm are given in [12, 13]. In [14, 15], the concept of maximal matching has been generalized as follows. Let  $G$  be a network, and for any node  $v$ , let  $\deg(v)$  denote its degree, that is, the number of edges incident to  $v$ . Let  $b(v)$  be a bound on the number of edges that can be incident to  $v$ , where  $0 \leq b(v) \leq \deg(v)$ . A subset  $M \subseteq E$  of edges is called a *generalized matching* or a *b-matching* of  $G$  if, for all nodes  $v \in V$ ,  $|E_v \cap M| \leq b(v)$ , where  $E_v$  denotes the set of edges incident to  $v$ . A *b-matching*  $M$  of  $G$  maximal if there does not exist another *b-matching*  $M'$  such that  $M' \supset M$ . Note that if  $b(v) = 1$  uniformly, then a *b-matching* is a simple matching. Our purpose in this paper is to develop a new distributed algorithm to maintain a *maximal b-matching* in a network that self-stabilizes in  $O(m)$  moves where  $m$  is the number of edges in the network. This analysis is independent of the function  $b(v)$ .

In a self-stabilizing algorithm, a node may change its local state by making a *move* (specification of an action). Algorithms are given as a set of rules of the form  $p(i) \Rightarrow M$ , where  $p(i)$  is a predicate and  $M$  is a move. A node  $i$  becomes *privileged* if  $p(i)$  is true. When a node becomes privileged, it may execute the corresponding move. We assume a serial model in which no two nodes move simultaneously. A central daemon selects, among all privileged nodes, the next node to move. If two or more nodes are privileged, one cannot predict which node will move next. Multiple protocols exist [16, 17, 18] that provide such a scheduler; hence our algorithms can be easily combined with any of those protocols to work under different schedulers as well.

## 2 Generalized Matching Protocol

Before we present our protocol for generalized matching, we briefly revisit the essential characteristics of the simple matching algorithm of [11]; the pseudocode is given in Figure 1. For each node  $i$ ,  $N(i)$  denotes its set of neighbors.

Each node  $i$  maintains a single variable whose value is either null, or points to a neighbor  $j \in N(i)$ . The algorithm has three rules: “A”, “P” and “W”: the edge between two adjacent nodes becomes part of a matching when each is pointing to the other. Rule **A** allows a node  $i$  to *accept* a proposed match with another node. Rule **P** allows a node  $i$  to *propose* to another node. Rule **W** allows a node  $i$  to *withdraw* a proposal.

Next, we consider the more general *b-matching*. One can design such an algorithm by providing each node with a *list* of pointers. By mimicking the rules of Algorithm 1 in a fairly straightforward way, one can obtain an algorithm which stabilizes in a polynomial number of moves, and which obtains a maximal *b-matching*. Our purpose in this paper is to develop a self-stabilizing algorithm for a *maximal b-matching* that runs in  $O(m)$  moves, independent of  $b$ .

We assume that the constant bound  $b(i)$  of node  $i$  is known to its neighbors. Every node  $i$  also maintains a list  $L(i)$  of pointers. We say  $L(i)$  is *valid* if  $0 \leq |L(i)| \leq b(i)$ ;  $L(i)$  contains only pointers to neighbors of  $i$ ; and it contains no duplicate references. It is possible that, at initialization, a list is not valid. However, it is easy to add a rule that forces a list to become valid. For this reason, to simplify the description of our algorithm, we assume that the lists are valid.

**Definition 1** Nodes  $i$  and  $j$  are matched if  $i \in L(j)$  and  $j \in L(i)$ , and we also say that the edge  $e = ij$  is matched. Upon stabilization, the *b-matching* is defined by the set  $M$  of matched edges.

**Definition 2** We define a set  $\text{Agr}(i)$  for each node  $i$  as follows

$$\text{Agr}(i) = \{j \in L(i) \mid i \in L(j)\}.$$

Figure 2 provides the complete pseudocode of our self-stabilizing maximal *b-matching* algorithm. Note that the rules are organized by precedence, and so a node that is eligible to execute Rule **A1** or Rule **A2** cannot execute **P**. A node that can execute Rule **P** cannot execute **W**. We make the following observations:

- Rules **A1** and **A2** allow a node to accept a proposal and create a permanent match with another node. Note that both **A1** and **A2** allow a node  $i$  to match with a neighbor  $j$  that is pointing to  $i$ . Rule **A1** simply adds  $j$  to  $L(i)$ , creating a match. Rule **A2**, however, replaces an unmatched entry in  $L(i)$  with  $j$ . Thus, **A1** increases the cardinality of both  $\text{Agr}(i)$  and  $L(i)$ , while **A2** increases the cardinality of  $\text{Agr}(i)$ , maintaining the size of  $L(i)$ .
- Rule **A2** is permitted only when  $|L(i)|$  is  $b(i)$  or  $b(i) - 1$ , and there exists some  $z \in L(i) - \text{Agr}(i)$ .

**Algorithm 1:**

**A:** if  $(i \rightarrow \text{null}) \wedge (\exists j \in N(i))(j \rightarrow i)$   
     **then** set  $(i \rightarrow j)$   
**P:** if  $(i \rightarrow \text{null}) \wedge (\forall k \in N(i))(\neg(k \rightarrow i)) \wedge (\exists j \in N(i))(j \rightarrow \text{null})$   
     **then** set  $(i \rightarrow j)$   
**W:** if  $(i \rightarrow j) \wedge (j \rightarrow k) \wedge (k \neq i)$   
     **then** set  $(i \rightarrow \text{null})$

**Figure 1. Maximal Matching Algorithm of Hsu and Huang**

- Rule **P** allows a node  $i$  to point to an adjacent node  $j$ , provided both  $i$  and  $j$  have an available space in their lists.
- The rule **W** allows a node to *withdraw* a proposal. If node  $i$  references node  $j$  whose list is full, and all of  $j$ 's pointers reference nodes other than  $i$ , then node  $i$  can remove  $j$  from its list.

We say that the system is *stable* if no node is able to make a move. The proof of our algorithm's correctness depends on showing that *i*) a stable system always exhibits a maximal  $b$ -matching, and *ii*) the system always becomes stable in a finite number of moves.

**Lemma 1** *When the system is stable,  $i \in L(j)$  if and only if  $j \in L(i)$ .*

**Proof:** Assume, by contradiction, that the system is stable and  $i \in L(j)$  but  $j \notin L(i)$ . Since  $i$  cannot make a move, the boolean condition for **A1** is false, and so  $|L(i)| \geq b(i) - 1$ . Since **A2**'s condition must be false, it follows that  $\text{Agr}(i) = L(i)$ . The list  $L(i)$  cannot have length  $b(i) - 1$  or **A1**'s condition would be true, so it follows that  $|\text{Agr}(i)| = |L(i)| = b(i)$ . It now follows that  $j$  can execute **W**, a contradiction.  $\square$

**Lemma 2** *When the system is stable, the set  $M$  of matched edges is a maximal  $b$ -matching.*

**Proof:** The set  $M$  is a  $b$ -matching because of Lemma 1 and the fact that the algorithm preserves the property  $|L(i)| \leq b(i)$ , for all  $i$ . To see that it is maximal, suppose that there exists another edge  $e = ij$ ,  $e \notin M$ , for which  $M \cup \{e\}$  is a  $b$ -matching. It must be that  $|L(i)| < b(i)$  and  $|L(j)| < b(j)$ . Since  $e \notin M$ , the condition of rule **P**, for node  $i$ , is true, a contradiction.  $\square$

**Lemma 3** *Once nodes  $i$  and  $j$  are matched, they remain so.*

**Proof:** No rule destroys a matched edge.  $\square$

Let  $m_1, m_2, \dots$ , be a sequence of moves made by Algorithm 2. In each of the moves that node  $i$  might make,

there is a *forcing neighbor*  $j$  which allows  $i$  to move. We use  $(i, j, X)$  to denote the execution of Rule  $X$  by node  $i$ , where  $j$  is its forcing neighbor. We will say that node  $i$  is *filled* if  $|L(i)| = b(i)$ , and *unfilled* otherwise. We say that  $i$  is *completely matched* if  $|\text{Agr}(i)| = |L(i)| = b(i)$ . It is possible that some node  $i$  does not become filled; but if it does,  $i$  will make no further moves.

**Lemma 4** *A completely matched node will never move again.*

**Proof:** This follows from the fact that a completely matched node  $i$  cannot move, and because of Lemma 3, no move by another node can destroy the condition of  $i$  being completely matched.  $\square$

**Lemma 5** *Suppose at time  $p$ , node  $j$  becomes filled (i.e., a move by  $j$  changes  $|L(j)|$  from  $b(j) - 1$  to  $b(j)$ ). Suppose also that at time  $p$ , there exists some node  $i \in N(j) - L(j)$  for which  $j \in L(i)$ . Then node  $j$  becomes completely matched.*

**Proof:** Since at time  $p$ ,  $|L(j)| = b(j) - 1$  and  $j \in L(i)$ , node  $j$  is eligible to accept a proposal, so the precedence of the rules precludes  $j$  from executing any rule except **A1** or **A2**. However **A2** does not cause  $|L(j)|$  to increase, so at time  $p$ , node  $j$  must have executed **A1**. But an inspection of rule **A1** shows that at time  $p$ ,  $L(j)$  becomes  $\text{Agr}(j)$ .  $\square$

Our most crucial lemma is the following.

**Lemma 6** *For each edge  $e = \{i, j\}$ , the move  $(i, j, \mathbf{W})$  occurs at most twice.*

**Proof:** Suppose  $m_s = (i, j, \mathbf{W})$  and  $m_t = (i, j, \mathbf{W})$ , where  $s < t$ . We will show that after time  $t$ , node  $j$  is completely matched. At time  $s + 1$ ,  $j \notin L(i)$ , and at time  $t$ ,  $j \in L(i)$ . Therefore, there must exist at least one move  $m_q = (i, j, \mathbf{P})$  for some  $q$ ,  $s < q < t$ . However, there can be several of these, since node  $i$  may remove node  $j$  using **A2** and then re-propose several times. Without loss of generality, assume that the last such proposal occurs at time  $q$ . Since  $|L(j)| < b(j)$  at time  $q$ , and since  $|L(j)| = b(j)$  at time  $t$ , there must

**Algorithm 2: Maximal  $b$ -matching()**

**A1:** if  $(\exists j \in N(i) - L(i))(i \in L(j)) \wedge ((|L(i)| < b(i) - 1) \vee (|\text{Agr}(i)| = |L(i)| = b(i) - 1))$   
**then** set  $L(i) = L(i) \cup \{j\}$   
**A2:** if  $(\exists j \in N(i) - L(i))(i \in L(j)) \wedge (|L(i)| \geq b(i) - 1) \wedge (\text{Agr}(i) \neq L(i))$   
**then** set  $L(i) = L(i) \cup \{j\} - \{z\}$  for some  $z \in L(i) - \text{Agr}(i)$   
**P:** if  $(|L(i)| < b(i)) \wedge (\exists j \in N(i) - L(i))(|L(j)| < b(j))$   
**then** set  $L(i) = L(i) \cup \{j\}$   
**W:** if  $(\exists j \in L(i))(|L(j)| = b(j)) \wedge (i \notin L(j))$   
**then** set  $L(i) = L(i) - \{j\}$

**Figure 2. Algorithm for Maximal  $b$ -matching**

be an intermediate move at say time  $p, q < p < t$ , at which  $|L(j)|$  becomes  $b(j)$ ; node  $j$  becomes filled. By Lemma 5, node  $j$  is then completely matched, and by Lemma 4, it cannot move again.  $\square$

**Lemma 7** *There are at most  $4m$  withdraw moves.*

**Proof:** By Lemma 6, for each edge  $e = \{i, j\}$ ,  $(i, j, \mathbf{W})$  occurs at most twice, and  $(j, i, \mathbf{W})$  occurs at most twice.  $\square$

Let us suppose that at time  $t$ , node  $i$  makes a proposal. We say that  $i$  *cancel*s the proposal if it later removes it from  $L(i)$  because of **A2**. We say that  $i$  *withdraw*s the proposal if it removes it from  $L(i)$  because of **W**. A *permanent* proposal is one that never is cancelled or withdrawn, and remains in  $L(i)$ . The number of withdrawn proposals is bounded by the number of withdraw moves, which is  $4m$ . Since every cancelled proposal always results in a matched edge, the number of cancelled proposals is bounded by  $m$ . The number of permanent proposals is bounded by the total capacity of all the lists, or  $2m$ . Thus, we have

**Lemma 8** *There are at most  $7m$  proposals.*

**Lemma 9** *Rules **A1** and **A2** together can be executed at most  $m$  times.*

**Proof:** Each produces a permanently matched edge.  $\square$

Lemma 7, Lemma 8, and Lemma 9 show that any execution of Algorithm 2 has at most  $12m$  moves. In light of Lemma 2, we have

**Theorem 1** *For any network and any function  $b$ , Algorithm 2 finds a maximal  $b$ -matching in  $O(m)$  moves.*

### 3 Conclusion

We have presented a simple self-stabilizing distributed algorithm to main a generalized matching in an arbitrary

network. We have shown that the algorithm stabilizes in at most  $O(m)$  moves where  $m$  denotes the number of edges in the network graph, i.e., the convergence time does not depend on the different  $b$  values at different nodes. It is interesting to note that when  $b(i) = 1$  for all  $i$ , our algorithm reduces to the simple matching algorithm (Algorithm 1) of [11].

### References

- [1] M Gradinariu and S Tixeuil. Self-stabilizing vertex coloration and arbitrary graphs. In *4th International Conference On Principles Of Distributed Systems, OPODIS'2000*, pages 55–70, 2000.
- [2] S Dolev and JL Welch. Crash resilient communication in dynamic networks. *IEEE Transactions on Computers*, 46:14–26, 1997.
- [3] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. McGraw Hill, 1998.
- [4] T.W. Haynes, S.T. Hedetniemi, and P.J. Slater. *Fundamentals of Domination in Graphs*. Marcel Dekker, 1998.
- [5] S. Fujita, T. Kameda, and M. Yamashita. A resource assignment problem on graphs. In *Proceedings of the 6th International Symposium on Algorithms and Computation*, pages 418–427, Cairns, Australia, December 1995.
- [6] S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Fault tolerant distributed coloring algorithms that stabilize in linear time. In *Proceedings of the IPDPS-2002 Workshop on Advances in Parallel and Distributed Computational Models*, pages 1–5, 2002.

- [7] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.
- [8] L. Lamport. Solved problems, unsolved problems, and non-problems in concurrency. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 1–11, 1984.
- [9] M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1):45–67, March 1993.
- [10] T. Herman. A comprehensive bibliography on self-stabilization, a working paper. *Chicago J. Theoretical Comput. Sci.*, <http://www.cs.uiowa.edu/ftp/selfstab/bibliography>.
- [11] S-C Hsu and S-T Huang. A self-stabilizing algorithm for maximal matching. *Information Processing Letters*, 43:77–81, 1992.
- [12] G. Tel. Maximal stabilizes in quadratic time. *Information Processing Letters*, 49(271–272), 1994.
- [13] S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Maximal matching stabilizes in time  $o(m)$ . *Information Processing Letters*, 80:221–223, 2001.
- [14] S. Goodman, S. T. Hedetniemi, and R. E. Tarjan. b-matchings in trees. *SIAM J. Comput.*, 5:104–108, 1976.
- [15] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, 1994.
- [16] M Nesterenko and A Arora. Stabilization-preserving atomicity refinement. In *DISC99 Distributed Computing 13th International Symposium, Springer-Verlag LNCS:1693*, pages 254–268, 1999.
- [17] G Antonoiu and PK Srimani. Mutual exclusion between neighboring nodes in an arbitrary system graph tree that stabilizes using read/write atomicity. In *Euro-par’99 Parallel Processing, Proceedings LNCS:1685*, pages 823–830, 1999.
- [18] J Beauquier, AK Datta, M Gradinariu, and F Magniette. Self-stabilizing local mutual exclusion and daemon refinement. In *DISC00 Distributed Computing 14th International Symposium, Springer-Verlag LNCS:1914*, 2000.