



ELSEVIER

Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

Information Processing Letters ●● (●●●) ●●-●●

**Information
Processing
Letters**

www.elsevier.com/locate/ipl

An anonymous self-stabilizing algorithm for 1-maximal independent set in trees

Zhengen Shi ^{*,1}, Wayne Goddard, Stephen T. Hedetniemi ²

Department of Computer Science, Clemson University, Clemson, SC 29634, USA

Received 29 September 2003; received in revised form 15 March 2004

Communicated by K. Iwama

Abstract

We present an anonymous, constant-space, self-stabilizing algorithm for finding a 1-maximal independent set in tree graphs (and some rings). We show that the algorithm converges in $O(n^2)$ moves under any central daemon (one that at each time-step selects one of the privileged nodes to move).

© 2004 Elsevier B.V. All rights reserved.

Keywords: Graph algorithms; Distributed computing; Fault tolerance; Maximal independent set

1. Introduction

Many services for a distributed system involve maintaining a global predicate over the entire network by using local information at each participating node. One approach to achieving this is self-stabilization, introduced by Dijkstra [3]. In the shared-variable version of this paradigm, every node executes the same set of self-stabilizing rules, and maintains and changes its own set of local variables based on the current values of its variables and those of its *neighbors*.

The basic idea is that the distributed system may be started in an arbitrary global configuration, but after

a finite interval the system reaches a correct global configuration, called a *legitimate configuration*. For a complete discussion of self-stabilization, see the books by Dolev [4] or Tel [13].

Several graph problems arise naturally in distributed systems. For example, distributed algorithms for finding matchings, independent sets, dominating sets and colorings have been studied [7,8,10,11]. Given the limited power of the self-stabilizing paradigm, one cannot expect to achieve optimality, and thus must settle for a good coloring, a maximal matching or a minimal dominating set.

In particular, in trying to identify a large subset of the nodes that are pairwise nonadjacent (i.e., an *independent set*), one can ask for a *maximal independent set* (every node not in the set is adjacent to a node in the set). Maximal independent sets are important for several distributed network applications (see, for example, Awerbuch et al. [1]), and several au-

* Corresponding author.

E-mail addresses: cshi@cs.clemson.edu (Z. Shi),
goddard@cs.clemson.edu (W. Goddard), hedet@cs.clemson.edu
(S.T. Hedetniemi).

¹ Research supported in part by NFS grant No. 0218495.

² Research supported in part by NFS grant No. 0218495.

thors have provided parallel or distributed algorithms (see, for example, Luby [9]) for this task. A simple self-stabilizing algorithm for this was found by both Shukla et al. [12] and Hedetniemi et al. [7].

However, a maximal independent set can be rather small: if, for example, the network has a star topology then the maximal independent set can be just the hub node. So we consider the problem of finding large independent sets. A *1-maximal independent set* [2] is a maximal independent set with the additional property that one cannot increase the cardinality of the independent set by removing one node and adding more nodes. For example, in the star topology, the single hub node does not constitute a 1-maximal independent set, since the node can be removed and replaced by all of its neighbors. Indeed, the only 1-maximal independent set in a star is the maximum independent set. We will show that a 1-maximal independent set in a tree always contains more than a third of the nodes (Lemma 1.1).

For special classes of networks, one can ask for optimal results. Thus, the general result of Ghosh et al. [6] shows how to find a maximum independent set in a tree even in the anonymous model (that is, it does not use IDs). However, the running time is (at least) cubic, and the storage at each node is not constant.

In this paper we provide a fast algorithm for finding a 1-maximal independent set in a tree that uses constant space at each node. The algorithm also works in some networks with cycles. In particular, it finds a 1-maximal independent set in a ring whenever the number of nodes is not a multiple of 3. It can be shown (Lemma 1.2) that if the network is a ring with number of nodes a multiple of 3, then no anonymous algorithm can ensure an independent set any bigger than the smallest maximal independent set (of cardinality $n/3$). Our algorithm is somewhat unusual in that it stabilizes on all graphs, but is only guaranteed to be correct on some graphs.

1.1. Notation and terminology

We define a distributed network as a connected, undirected graph G with node set V and edge set E . Two nodes joined by an edge are said to be *neighbors*. We use $N(i)$ to denote the set of neighbors of node i —its (open) *neighborhood*.

A self-stabilizing algorithm is presented as a set of rules, each with a boolean predicate and an action. A node is said to be *privileged* if the predicate is true. If a node becomes privileged, it may execute the corresponding action: we call this a *move*. We assume that there exists a *central daemon* [3,8], which at each time-step selects one of the privileged nodes to move (and thus two nodes never move at the same time).

When no further move is possible, we say that the system is in a *stable configuration*. While the definition of self-stabilizing is normally more general, since this is a graph algorithm we say that an algorithm is self-stabilizing if from any initial configuration it always terminates in a legitimate stable configuration after a finite number of moves no matter the selections of the daemon.

1.2. Goodness

We observe that every 1-maximal independent set in a tree contains more than one third of the nodes.

Lemma 1.1. *For any tree T of order n , every 1-maximal independent set has cardinality at least $(n + 1)/3$, and this result is best possible.*

Proof. By induction on n . If the diameter of the tree is 1 or 2, then the tree is a star, and the cardinality of any 1-maximal independent set is $n - 1$, which is at least $(n + 1)/3$. So assume the diameter is at least 3.

Consider a diametrical path $abcd\dots$ and a 1-maximal independent set S . There are two cases. If the node b is not in S , then all its leaf neighbors are in S . Let subtree T' be the component of $T - bc$ that contains c . The restriction of S to T' is 1-maximal. Thus, if b has degree x , it follows that $|S| \geq (x - 1) + (n - x + 1)/3 > (n + 1)/3$ (since $x \geq 2$).

So assume for all diametral paths, the second node is in S . It follows that b has degree 2 (else it can be replaced by its leaf neighbors). Further, every neighbor of c , apart from possibly one node, is either a leaf node or has degree 2 and is adjacent to a leaf node. Let subtree T' be the component of $T - cd$ that contains d . The restriction of S to T' is 1-maximal. If c has degree y , it follows that $|S| \geq (y - 1) + (n - (2y - 1) + 1)/3 = n/3 + (y - 1)/3 \geq (n + 1)/3$.

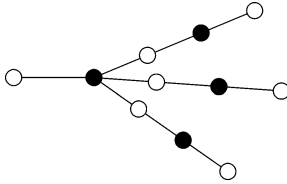


Fig. 1. A tree with the smallest 1-maximal independent set.

That this result is best possible follows from the tree shown in Fig. 1: a star in which each edge except one is subdivided twice. \square

We observe that any anonymous self-stabilizing algorithms cannot produce a 1-maximal independent set on a ring with order a multiple of 3.

Lemma 1.2. *Let \mathcal{A} be an anonymous self-stabilizing algorithm that produces an independent set S in any ring. Then for any ring of order n where n is a multiple of 3, the algorithm \mathcal{A} cannot guarantee that $|S| \geq n/3$.*

Proof. Since algorithm \mathcal{A} works on any ring, it stabilizes on a ring with 3 processors. Suppose the final states are X, Y and Z , where X implies the node is in the independent set S . Then the 6-cycle with states X, Y, Z, X, Y, Z is also stable, and has only two nodes in S . A similar argument holds for any ring which is a multiple of 3. \square

2. Description of algorithm

In the algorithm for 1-maximal independent set, each node is in one of a finite number of distinct states. Those nodes in the state called 0 will end up being in the desired set: let us call this set \mathcal{M} . A node with no neighbor in state 0 will change to state 0 and a node in state 0 with a neighbor in state 0 will change to something else. This idea readily produces a maximal independent set.

To achieve 1-maximality, however, a node must be able to leave \mathcal{M} when that would allow two of its neighbors to enter \mathcal{M} . Available neighbors are those which have no other neighbor in state 0: these will be in the state called 1. In order to allow this *interchange*, we implement a hand-shaking process: the node offers to leave \mathcal{M} by changing to state $0'$, the relevant neighbors agree to enter \mathcal{M} by changing

to state $1'$, the node leaves by changing to state $2'$, and then the relevant neighbors go in by changing to state 0.

Specifically, the set of states is $\{0, 0', 1, 1', 2, 2'\}$. The states with a prime are *transition states*, used in the hand-shaking process described above. These transition states will be absent when the algorithm terminates if the network satisfies certain properties (Lemma 3.2).

For the purposes of the algorithm, the nodes with state $0'$ are also considered to be in \mathcal{M} . Then the states 0, 1 and 2 are used to indicate that a node has zero, one, or at least two neighbors in \mathcal{M} , respectively.

The basic approach of the algorithm can be summarized as follows. A node does the following:

- If not involved in a transition process, then set state to the number of neighbors in state 0 or state $0'$. (The value 2 is used to indicate two or more such neighbors.)
- If in state 0 and adjacent to at least two 1s, change state to $0'$.
If in state 1 and adjacent to a $0'$, change state to $1'$.
If in state $0'$ and adjacent to at least two $1'$ s, change state to $2'$.
If in state $1'$ and adjacent to no $0'$, change state to 0.
If in state $2'$ and adjacent to no $1'$, change state to 2.

The complexity of the actual algorithm arises from invalid initial states and from two interchanges affecting one another.

2.1. The rules

Definition. For a state y , we use the notation $\mathcal{S}(y)$ to represent the set of nodes in state y . Furthermore, we use the notation $\mathcal{S}(y_1/y_2/y_3/\dots)$ to denote $\mathcal{S}(y_1) \cup \mathcal{S}(y_2) \cup \mathcal{S}(y_3) \cup \dots$.

For example, the notation $\mathcal{S}(0)$ denotes the nodes in state 0.

The state of a node is stored in a local variable denoted x . The states with a prime are transition states. We will also identify the prime with a virtual flag—we will say the flag is set when the node is in a transition

state, and clearing the flag will mean changing from state i' to state i .

To define the rules of the algorithm, we define the following function f .

Definition. Let i be a node and t a state. Then we define:

$$f_i(t) = \min\{2, |N(i) \cap \mathcal{S}(t)|\}.$$

The function f_i (originally used in [5]) gives the number of neighbors of node i in a specified state. We further use the notation:

$$f_i(x/y/z/\dots) = \min\{2, f_i(x) + f_i(y) + f_i(z) + \dots\}.$$

When the node i is clear from context, we will simply write f rather than f_i .

We also utilize the concept of *bad edge*, which is defined next. The rules will be such that a bad edge can only occur as a result of faulty initialization.

Definition. A bad edge is an edge connecting two nodes in the following list of pairs of states: 0-0; 0-0'; 0'-0'; 0'-2'; 1'-1'; 2'-2'.

The full algorithm for finding a 1-maximal independent set is given as Algorithm ONEMAX.

3. Correctness

The validity of a self-stabilizing algorithm depends upon two conditions: correctness and convergence. Correctness is that every stable configuration it can reach must be legitimate; that is, every stable configuration has the desired global property. This condition is the main focus of the lemmas in this section. Convergence is that the algorithm must always reach a stable configuration after a finite number of moves. The second condition is analyzed in the next section in which the property of convergence is proved and the complexity is bounded.

Lemma 3.1. *The following holds in any stable configuration produced by this algorithm:*

- (1) *There is no node in state 2'.*
- (2) *If there is a node in state 0', it must be adjacent to a node in state 1.*
- (3) *If there is a node in state 1 and it is adjacent to a node in state 0', then it must also be adjacent to a node in state 1'.*
- (4) *If there is a node in state 1', it must be adjacent to a node in state 0'.*

Proof. Consider any stable configuration.

Algorithm ONEMAX

All moves are tried in the listed order

V1: **if** flag is set **and** node is incident on bad edge

and after clearing flag node would not be incident on any bad edge

then clear flag

V2: **if** flag is clear **and** $x \neq f(0/0')$ **and** $(f(0/0') \geq 1 \text{ or } f(1'/2') = 0)$

then set $x = f(0/0')$

C1: **if** $x = 0$ **and** $f(1) = 2$ **and** $f(0/0'/2') = 0$

then set $x = 0'$

C2: **if** $x = 0'$ **and** $(f(1/1') \leq 1 \text{ or } f(0/0') \geq 1)$

then set $x = f(0/0')$

C3: **if** $x = 0'$ **and** $f(1') = 2$ **and** $f(0'/2') = 0$

then set $x = 2'$

C4: **if** $x = 2'$ **and** $f(1') = 0$

then set $x = f(0/0')$

C5: **if** $x = 1$ **and** $f(0') = 1$ **and** $f(0/1'/2') = 0$

then set $x = 1'$

C6: **if** $x = 1'$ **and** $(f(0') \neq 1 \text{ or } f(0/1'/2') \geq 1)$

then set $x = f(0/0')$

- (1) Suppose there is a node $v \in \mathcal{S}(2')$. Since v is not privileged for Rule C4, $f_v(1') \geq 1$.
Let u be a neighbor of v in state $1'$. Then $f_u(0/1'/2') \geq 1$. Hence, u is privileged for Rule C6, a contradiction.
- (2) Assume there is a node $v \in \mathcal{S}(0')$. Since v is not privileged for Rule C2, it must be that $f_v(1/1') = 2$ and $f_v(0/0') = 0$. From part (1), $\mathcal{S}(2') = \emptyset$. So, since v is not privileged for Rule C3, it must be that $f_v(1') \leq 1$. Hence, $f_v(1) \geq 1$.
- (3) Suppose there is a node $v \in \mathcal{S}(1)$ that is adjacent to a node in state $0'$, but not adjacent to any node of state $1'$. By part (1), $\mathcal{S}(2') = \emptyset$. Since v is not privileged for Rule V2, we have $f_v(0/0') = 1$. From the assumption, we have $f_v(0') \geq 1$. Thus $f_v(0') = 1$ and $f_v(0/1'/2') = 0$. Hence, v is privileged for Rule C5, a contradiction.
- (4) Assume there is a node $v \in \mathcal{S}(1')$. Since v is not privileged for Rule C6, $f_v(0') = 1$. \square

Lemma 3.2. *If the length of no cycle in the network is divisible by 3, then any stable configuration contains no node of transition state.*

Proof. Consider any stable configuration.

From Lemma 3.1, we know that $\mathcal{S}(2') = \emptyset$. We also know that if a node in state $0'$ exists, it must be adjacent to at least one node of state 1. If a node in state 1 is adjacent to a node in state $0'$, it must also be adjacent to at least one node of state $1'$. If a node in state $1'$ exists, it must be adjacent to a node of state $0'$.

Thus, if a node in state $0'$ or $1'$ exists in a stable state, by applying the above, we can obtain an arbitrarily long walk of pattern "...0'11'0'11'0'11'...". Since graph G is finite, this walk contains a cycle.

Because the repeating portion of the pattern is of length 3, the length of the cycle must be divisible by 3. Thus, if the length of every cycle in G is not divisible by 3, there is no node of transition state in a stable configuration. \square

Corollary 3.1. *In a stable configuration in a tree, there is no node in a transition state.*

Lemma 3.3. *In a stable configuration, the set $\mathcal{S}(0/0')$ constitutes an independent set.*

Proof. If $\mathcal{S}(0/0')$ is not an independent set, then there exist a node $i \in \mathcal{S}(0/0')$ with a neighbor in $\mathcal{S}(0/0')$. If node i is in state 0, then its flag is clear and $x_i \neq f_i(0/0')$ and $f_i(0/0') \geq 1$. So i is privileged for Rule V2. If i is in state $0'$, then $f_i(0/0') \geq 1$ and so it is privileged for Rule C2. Both cases cause a contradiction. \square

Lemma 3.4. *In a stable configuration, if there is no node in transition state, then $\mathcal{S}(0)$ constitutes a 1-maximal independent set.*

Proof. By Lemma 3.3, $\mathcal{S}(0)$ is independent. If $\mathcal{S}(0)$ is not a maximal independent set, there must exist a node $v \in \mathcal{S}(1/2)$ that is adjacent to no node in state 0. For node v , its flag is clear and $x_v \neq f_v(0/0')$ and $f_v(1'/2') = 0$, and so it is privileged for Rule V2, a contradiction.

If $\mathcal{S}(0)$ is not a 1-maximal independent set, there must exist a node $w \in \mathcal{S}(0)$ with two or more neighbors in $\mathcal{S}(1)$. Thus, $f_w(1) = 2$ and $f_w(0/0'/2') = 0$ and so node w is privileged for Rule C1, a contradiction. \square

Corollary 3.2. *In a stable configuration in a tree, $\mathcal{S}(0)$ constitutes a 1-maximal independent set.*

Finally we observe that the algorithm solves the original problem:

Lemma 3.5. *Every 1-maximal independent set is achievable as $\mathcal{S}(0)$ in a stable configuration.*

Proof. Let T be a 1-maximal independent set. For every node v in the network, set its x -value to $|N(v) \cap T|$. Then since T is maximal independent, $\mathcal{S}(0) = T$. The only rule that a node might be privileged for is Rule C1. But since T is 1-maximal, this is avoided. \square

4. Convergence and complexity analysis

To prove the algorithm always reaches a stable configuration after a finite number of moves, we define the following concepts.

Definition. A clean node is one that has moved at least once. Otherwise it is dirty.

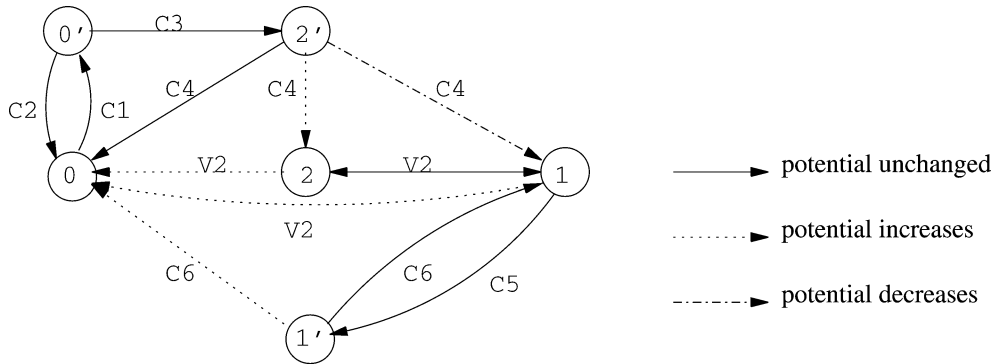


Fig. 2. The clean moves.

Lemma 4.1. A clean node is not incident on any bad edge.

Proof. Rule V1 explicitly requires that the node is not incident on any bad edge after clearing the flag. Clearly, the predicates of the other rules guarantee that the node is not incident on any bad edge after the move. For example, if x becomes 0, then the node is not adjacent to any node in state 0 or 0'. \square

Lemma 4.2. There are twelve possible moves by a clean node as shown in Fig. 2.

Proof. A clean node in state 0 or 0' cannot have a neighbor in state 0 or 0', and hence cannot move to state 1 or 2.

Consider a clean node v in state 1'. Since v is clean, it last executed Rule C5. At that point, $f_v(0/1'/2') = 0$ and $f_v(0') = 1$. By Rule C5, a neighbor in state 1 cannot move to state 1'. By Rule V2, a neighbor in state 1 or 2 cannot move to state 0. Thus, when v executes Rule C6, the only possible move, the neighbor that was in state 0' earlier is now in state 2' or state 0. \square

To bound the number of moves, we introduce a potential function.

Definition. For a graph G , let $\Phi_1 = |\mathcal{S}(0/0'/2')|$ and let Φ_2 be the number of edges that connect one node in state 0 to the other node in state 2'. The potential function of the system Φ is defined from Φ_1 and Φ_2 :

$$\Phi = 3\Phi_1 - 2\Phi_2.$$

Lemma 4.3. Consider a move by a clean node. The moves that increase the potential Φ are $1 \rightarrow 0$, $2 \rightarrow 0$, $1' \rightarrow 0$, and $2' \rightarrow 2$. The move that decreases the potential is $2' \rightarrow 1$. All other moves do not affect the potential (see Fig. 2).

Proof. Let v be the clean node. We consider each possible move by v in turn.

(1) State 1 or 2 to state 0.

By Rule V2, a node in state 1 or 2 does not change to state 0 if it is adjacent to a node in state 2'. Thus, the change of the potential function is $\Delta\Phi = 3\Delta\Phi_1 - 2\Delta\Phi_2 = 3(1) - 2(0) = 3$.

(2) State 0' or 2' to state 0.

A clean node in state 0' or 2' is not adjacent to a node in state 2'. Thus, $\Delta\Phi = 3(0) - 2(0) = 0$.

(3) State 0 to state 0'.

An edge connecting nodes in states 0' and 2' is bad, and so v is not adjacent to a node in state 2'. Thus, $\Delta\Phi = 3(0) - 2(0) = 0$.

(4) State 0' to state 2'.

An edge connecting nodes in states 0' and 0 is bad, and so v is not adjacent to a node in state 0. Thus, $\Delta\Phi = 3(0) - 2(0) = 0$.

(5) State 2' to state 2.

Since v is clean, it has no neighbors in state 0'. So it has at least two neighbors in state 0. Thus, $\Delta\Phi \geq 3(-1) - 2(-2) = 1$.

(6) State 1' to state 0.

By the argument in the proof of the previous lemma, when v executes Rule C6 and changes to 0, only one neighbor can be (and in fact is) in state 2': the neighbor that was in state 0' earlier. Thus, $\Delta\Phi = 3(1) - 2(1) = 1$.

(7) State 2' changes to state 1.

The node in state $2'$ is adjacent to one node in state 0. Thus, $\Delta\Phi = 3(-1) - 2(-1) = -1$.

All other moves clearly do not affect the potential. \square

Lemma 4.4. *A clean node can change from state $2'$ to state 1 or 0 only once.*

Proof. When the node v enters state $2'$, it is adjacent to two nodes with state $1'$. If these are both clean, then by the above they can only change to state 0 (and by Rule C1 are not privileged until v moves). Hence, a clean v can change from $2'$ to 1 or 0 only if one or more of its neighbors with state $1'$ were dirty, but at the time of the move these dirty neighbors have changed from $1'$ and are now clean. \square

If a new node joins or an existing node leaves the set $\mathcal{S}(0/0')$, we say the set $\mathcal{S}(0/0')$ changes. Note that the set is not considered changed if a node moves between the sets $\mathcal{S}(0)$ and $\mathcal{S}(0')$.

Lemma 4.5. *The set $\mathcal{S}(0/0')$ changes at most $O(m)$ times, where m is the number of edges in the network.*

Proof. The potential Φ is bounded from below by $-2m$ and from above by $3n$. From Fig. 2, a node in state 0 or $0'$ must go through state $2'$ to leave the set $\mathcal{S}(0/0')$. By Lemma 4.4, a clean node can change from state $2'$ to state 1 or 0 only once. The move from state $2'$ to 2 will increase the potential and is bounded by $O(m)$. \square

Lemma 4.6. *Assume $\mathcal{S}(0/0')$ is fixed, there is $O(n)$ moves.*

Proof. Since $\mathcal{S}(0/0')$ is fixed, there is at most one move between states 1 and 2 per node. Thus, the number of moves between states 0 and $0'$ is constant. Hence the number of moves between 1 and $1'$ is constant. There is at most one move of a node in states $2'$ per node. \square

Lemma 4.7. *The algorithm converges in $O(mn)$ time.*

Proof. This follows directly from Lemmas 4.5 and 4.6. \square

Corollary 4.1. *The algorithm ONEMAX stabilizes to a 1-maximal independent set in $O(n^2)$ steps in an arbitrary tree.*

References

- [1] B. Awerbuch, A.V. Goldberg, M. Luby, S.A. Plotkin, Network decomposition and locality in distributed computation, in: Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 364–369.
- [2] B. Bollobás, E.J. Cockayne, C.M. Mynhardt, On generalized minimal domination parameters for paths, *Discrete Math.* 86 (1–3) (1990) 89–97.
- [3] E.W. Dijkstra, Self-stabilizing systems in spite of distributed control, *Comm. ACM* 17 (11) (January 1974) 643–644.
- [4] S. Dolev, *Self-Stabilization*, MIT Press, Cambridge, MA, 2000.
- [5] G.S. Domke, Variations of colorings, coverings and packings of graphs, PhD thesis, Clemson University, Clemson, SC, 1988.
- [6] S. Ghosh, A. Gupta, M. Karaata, S. Pemmaraju, Self-stabilizing dynamic programming algorithms on trees, in: Proceedings of the Second Workshop on Self-Stabilizing Systems, 1995, pp. 11.1–11.15.
- [7] S.M. Hedetniemi, S.T. Hedetniemi, D.P. Jacobs, P.K. Srimani, Self-stabilizing algorithms for minimal dominating sets and maximal independent sets, *Comput. Math. Appl.* 46 (5–6) (2003) 805–811.
- [8] S.-C. Hsu, S.-T. Huang, A self-stabilizing algorithm for maximal matching, *Inform. Process. Lett.* 43 (1992) 77–81.
- [9] M. Luby, A simple parallel algorithm for the maximal independent set problem, *SIAM J. Comput.* 15 (4) (1986) 1036–1053.
- [10] A. Panconesi, A. Srinivasan, The local nature of δ -coloring and its algorithmic applications, *Combinatorica* 15 (1995) 225–280.
- [11] S. Rajagopalan, V. Vazirani, Primal-dual RNC approximation algorithms for set cover and covering integer programs, *SIAM J. Comput.* 28 (1998) 525–540.
- [12] S. Shukla, D. Rosenkrantz, S. Ravi, Observations on self-stabilizing graph algorithms for anonymous networks, in: Proceedings of the Second Workshop on Self-Stabilizing Systems, 1995, p. 7.17.15.
- [13] G. Tel, *Introduction to Distributed Algorithms*, second ed., Cambridge Univ. Press, Cambridge, UK, 2000.