

## SELF-STABILIZING MAXIMAL $k$ -DEPENDENT SETS IN LINEAR TIME

MARTIN GAIRING, WAYNE GODDARD, STEPHEN T. HEDETNIEMI, DAVID P. JACOBS\*

*Department of Computer Science, Clemson University  
Clemson, SC 29634-0974, USA*

Received (received date)

Revised (revised date)

Communicated by (Name of Editor)

### ABSTRACT

In a graph or network  $G = (V, E)$ , a set  $S \subseteq V$  is  $k$ -dependent if no node in  $S$  has more than  $k$  neighbors in  $S$ . We show that for each  $k \geq 0$  there is a self-stabilizing algorithm that identifies a maximal  $k$ -dependent set, and stabilizes in  $O(kn)$  moves, where  $n$  is the number of nodes. An interesting by-product of our paper is the new result that in any graph there exists a set that is both maximal  $k$ -dependent and minimal  $(k + 1)$ -dominating. The set selected by our algorithm, in fact, has this property.

*Keywords:* self-stabilizing, algorithm,  $k$ -dependent,  $k$ -dominating

### 1. Introduction

A distributed system can be modeled by a connected, undirected graph  $G = (V, E)$  with node set  $V$  and edge set  $E$ . If  $i \in V$ , then  $N(i)$ , its *open neighborhood*, denotes the set of nodes to which  $i$  is adjacent. Every node  $j \in N(i)$  is called a *neighbor* of node  $i$ . We let  $n$  and  $m$  denote the cardinalities of  $V$  and  $E$ , respectively.

Self-stabilization is a paradigm for distributed systems that allows a system to achieve a desired global state, even in the presence of faults. The concept was introduced in 1974 by Dijkstra [2], but serious work on self-stabilizing algorithms did not start until the late 1980's. (See [16, Ch. 15] for an introduction to self-stabilizing algorithms.) In a self-stabilizing algorithm, each node maintains its own set of local variables, and can make decisions based on the contents of any neighbor's local variables. The contents of a node's local variables determine its *local state*. The system's *global state* is the union of all local states.

A node may change its local state by making a *move*. A move consists of instructions for changing the values of one or more variables of the node. Algorithms are given as a set of rules of the form **if**  $p(i)$  **then**  $M$ , where  $p(i)$  is a predicate and  $M$  is a move. Normally, self-stabilizing algorithms are written so that, for a given node, at most one of its predicates is true, in which case we say the node  $i$  becomes *privileged*. When a node becomes privileged, it may execute the corresponding

---

\*Corresponding author.

move. We assume there exists a daemon, an adversary oracle, as introduced in [2], which selects one of the privileged nodes to move. Following the serial model, we assume that no two nodes move at the same time. When no further nodes are privileged, we say that the system is *stable* or is in a stable state. A self-stabilizing algorithm is correct if:

- 1) every stable state is legitimate, that is, has the desired global property, and
- 2) from any initial state, it always reaches a stable state after a finite number of moves.

Notice that it is quite possible for a correct self-stabilizing algorithm to reach a legitimate state which is not stable.

Problems that can be solved by a straightforward greedy method in the conventional algorithmic model often require a far more clever approach in the self-stabilizing model. For example, finding a maximal matching (i.e. a set of disjoint edges that cover all remaining edges) in a graph is trivial: starting with an empty set of edges, keep adding another disjoint edge to the set as long as one exists. To describe and prove the correctness of a self-stabilizing algorithm for this same problem takes considerably more effort [12,13,15].

A set  $S \subseteq V$  is *k-dependent* if and only if  $\forall i \in S, |N(i) \cap S| \leq k$ . A *k-dependent* set not properly contained in any other *k-dependent* set is said to be *maximal k-dependent*. A set  $S \subseteq V$  is *k-dominating* if and only if  $\forall i \in V - S, |N(i) \cap S| \geq k$ . Such a set not properly containing any other *k-dominating* set is said to be *minimal k-dominating*. In the context of distributed systems, a *k-dominating* set represents a situation in which every node either has a given resource, or if not, has at least *k* neighbors, each of whom has such the resource. Similarly, a *k-dependent* set represents a situation in which closeness of resources is generally not desired, and we set a limit, say *k*, on the number of resources that are close.

The concepts of *k-dependent* sets and *k-dominating* sets were first defined by Fink and Jacobsen [8,9] as generalizations of independent sets and dominating sets. Recall that a set  $S \subseteq V$  is called *independent* if no two vertices in *S* are adjacent, and is a *dominating* set if every vertex in  $V - S$  is adjacent to at least one vertex in *S*. Thus, by design, independent sets are 0-dependent sets, and dominating sets are 1-dominating sets. (Furthermore, 1-dependent sets consist of independent vertices and edges, and 2-dependent sets consist of independent paths and cycles.)

Jacobson and Peters [14] showed that determining the *maximum* cardinality of a *k-dependent* set is NP-hard and also exhibited linear-time algorithms for finding this parameter if the graph is restricted to trees and series-parallel graphs. Djidjev, Garrido, Levkopoulos and Lingas [4] also presented a linear time algorithm to find the parameter for trees, proved a tight lower bound for this number, presented an  $O(\log n)$  parallel implementation, and showed that for graphs of bounded tree-width, the problem has a linear time solution. The papers by Diks, Garrido and Lingas [3] and by Dessmark, Jansen and Lingas [1] also give algorithms for various classes of graphs and parallel implementations. Further properties of *k-dependent* sets are found in [5,6,7,10].

*The purpose of this paper is to give an  $O(kn)$  self-stabilizing algorithm that finds a maximal k-dependent set in any graph.* An interesting outcome of this algorithm

is that the set it obtains is also minimal  $(k+1)$ -dominating. That such a set always exists, for any  $k \geq 0$ , was not known.

## 2. Self-stabilizing algorithm for maximal $k$ -dependent sets

We will show that Algorithm 2.1 finds a maximal  $k$ -dependent set  $S$ . Each node  $i$  has single variable  $f(i) \in \{0, 1\}$ . We let  $f(S) = \sum_{v \in S} f(v)$ , for any set  $S \subseteq V$ . The algorithm is identically stored and executed in each node  $i$ . Like any self-stabilizing algorithm, we do not specify initial values for the variables because the algorithm is expected to run correctly from *any* initial state.

---

**Algorithm 2.1:** MAXIMAL  $k$ -DEPENDENT SET ()

---

**local**  $f \in \{0, 1\}$ ;

**R1** :if  $f(i) = 0 \wedge f(N(i)) \leq k$   
**then**  $f(i) = 1$ ;

**R2** :if  $f(i) = 1 \wedge f(N(i)) > k$   
**then**  $f(i) = 0$ ;

---

Algorithm 2.1 is a generalization of an  $O(n)$  algorithm by Hedetniemi et. al. for finding a maximal independent set [11]. That algorithm can be seen as a special case of the present algorithm for  $k = 0$ . We will say that Algorithm 2.1 is in a *legitimate* state if  $S = \{i | f(i) = 1\}$  is a maximal  $k$ -dependent set. Since  $k$ -dependence is a property that is preserved by subsets, we have

**Lemma 1** *A  $k$ -dependent set  $S$  is maximal  $k$ -dependent if and only if  $\forall u \in V - S$ ,  $S \cup \{u\}$  is not  $k$ -dependent.*

To prove that Algorithm 2.1 is correct, we must show that it satisfies both requirements in the previous section. The following lemma shows that Algorithm 2.1 satisfies the first of these.

**Lemma 2** *If Algorithm 2.1 reaches a stable state, then  $S = \{i | f(i) = 1\}$  is maximal  $k$ -dependent.*

**Proof:** If  $S$  is not a  $k$ -dependent set, then there must exist a node  $i \in S$  having more than  $k$  neighbors in  $S$ , i.e.  $\exists i \in S$  such that  $|N(i) \cap S| > k$ . But then **R2** can be executed by  $i$ . On the other hand, if  $S$  is  $k$ -dependent but not maximal, then by Lemma 1 there exists a node  $i \in V - S$  (i.e.  $f(i) = 0$ ) such that  $S \cup \{i\}$  is still  $k$ -dependent. Therefore  $i$  has at most  $k$  neighbors in  $S$  and can execute **R1**.  $\square$

We now prove that Algorithm 2.1 meets the second correctness requirement, namely that it always achieves stabilization. The execution of our algorithm may be represented as a sequence of moves  $M_1, M_2, \dots$ , in which  $M_t$  denotes the  $t$ -th

move. The system's initial state is denoted by  $S_0$ , and for  $t > 0$  the state resulting from  $M_t$  is denoted by  $S_t$ . In state  $S_t$ , let

$$S = \{i | f(i) = 1\},$$

and let  $F$  be the subset of  $S$  consisting of those nodes that have made at least one move. That is,  $f_t$  excludes those members of  $S$  that are still positive from initialization. We let  $f_t$  denote the cardinality of  $F$ . Now let

$$P = \{e \in E | e = ij \wedge i \in S \wedge j \in S \wedge (i \in F \vee j \in F)\},$$

and let  $p_t$  denote the cardinality of  $P$ . That is,  $p_t$  is the number of edges having both endpoints with positive value, with at least one endpoint in  $F$ .

In [11] it was shown that the number of moves that Algorithm 2.1 can make for  $k = 0$  is bounded by  $2n$ . This follows from the fact, that for  $k = 0$ , a single node can never make more than two moves. For  $k \geq 1$  this is no longer true. We will show stabilization using a variant function whose value at time  $t$  is defined as

$$\Psi_t = k f_t - p_t.$$

The following two lemmas are straightforward.

**Lemma 3** *If move  $M_t$  uses rule **R1**, then*

- i**  $f_{t+1} = f_t + 1$  and
- ii**  $p_t \leq p_{t+1} \leq p_t + k$ .

**Lemma 4** *If a node  $i \in F$  uses rule **R2** at time  $t$ , then*

- i**  $f_{t+1} = f_t - 1$  and
- ii**  $p_{t+1} \leq p_t - (k + 1)$ .

**Lemma 5** *An **R1** move does not cause  $\Psi_t$  to decrease.*

**Proof:** From Lemma 3 we have

$$\Psi_{t+1} = k f_{t+1} - p_{t+1} = k(f_t + 1) - p_{t+1} \geq k(f_t + 1) - (p_t + k) = \Psi_t. \square$$

**Lemma 6** *An **R2** move by a node  $i \in F$ , causes  $\Psi_t$  to increase by at least one.*

**Proof:** From Lemma 4 we have:

$$\Psi_{t+1} = k f_{t+1} - p_{t+1} = k(f_t - 1) - p_{t+1} \geq k(f_t - 1) - (p_t - (k + 1)) = \Psi_t + 1. \square$$

**Lemma 7** *An **R2** move by a node in  $i \in S - F$  does not cause  $\Psi_t$  to decrease.*

**Proof:** Since  $i \notin F$ ,  $f_{t+1} = f_t$ . If no neighbors of  $i$  belong to  $F$  then we will also have  $p_{t+1} = p_t$ , and so  $\Psi_{t+1} = \Psi_t$ . On the other hand, if there exist neighbors of  $i$  that belong to  $F$  at time  $t$ , the size of  $P$  will decrease, and so  $\Psi$  will increase.  $\square$

**Lemma 8** For each  $t$ ,  $\Psi_t \leq \Psi_{t+1}$ .

**Proof:** This follows from Lemma 5, Lemma 6, and Lemma 7.  $\square$

Since  $\Psi_0 = 0$ , by the previous lemma, at any time  $t$

$$0 \leq \Psi_t \leq kn. \tag{1}$$

**Lemma 9** At most  $n + kn$  **R2** moves are possible.

**Proof:** Each node can make at most one **R2** move when it is not a member of  $F$ , so at most  $n$  such **R2** moves are possible. The remaining **R2** moves, by Lemma 6, cause  $\Psi_t$  to increase. Thus, from Lemma 8 and (1), it is clear that there can only be at most  $kn$  such moves.  $\square$

**Lemma 10** At most  $2n + kn$  **R1** moves are possible.

**Proof:** This follows from Lemma 9, and from the fact that **R1** moves and **R2** moves have to alternate at each node. Thus, at each node, the numbers of **R1** moves and **R2** moves can differ by at most one.  $\square$

**Theorem 1** Algorithm 2.1 finds a maximal  $k$ -dependent set, and stabilizes within  $2kn + 3n$  moves.

**Proof:** This follows from Lemma 2, Lemma 9, and Lemma 10.  $\square$

Since a superset of a  $k$ -dominating set is  $k$ -dominating, we have

**Lemma 11** A  $k$ -dominating set  $S$  is minimal  $k$ -dominating if and only if  $\forall u \in S$ ,  $S - \{u\}$  is not  $k$ -dominating.

**Lemma 12** When Algorithm 2.1 stabilizes, the set  $S = \{i | f(i) = 1\}$  is also a minimal  $(k + 1)$ -dominating set.

**Proof:** When Algorithm 2.1 terminates then because of **R1**,  $\forall i \in \{V - S\}$  we have  $f(N(i)) \geq k + 1$ . Thus,  $S$  is a  $(k + 1)$ -dominating set. We claim it is also minimal. If not, then by Lemma 11 there exist a node  $i \in S$  for which  $S - \{i\}$  is a  $(k + 1)$ -dominating set. Therefore,  $i$  has at least  $k + 1$  neighbors in  $S$  and can execute rule **R2**.  $\square$

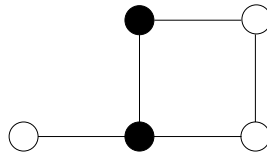


Fig. 1. A maximal 1-dependent set that is not 2-dominating.

When  $k = 0$ , every maximal 0-dependent set is always minimal 1-dominating. However, this is not always true for  $k > 0$ . For example the set  $S$  in Figure 1 is

maximal 1-dependent but not minimal 2-dominating. But a corollary to Lemma 12 and Theorem 1 is

**Theorem 2** *For every  $k \geq 0$  and every graph  $G$ , there exists a vertex set that is both maximal  $k$ -dependent and minimal  $(k + 1)$ -dominating.*

### 3. Example

Theorem 1 says that when  $k = 1$ , Algorithm 2.1 stabilizes within  $5n$  moves. We now give an infinite family of trees in which there is an execution lasting exactly  $2n - \lceil \log_2 n \rceil$  moves. Our example also shows that, unlike the case of  $k = 0$  as given in [11], for  $k = 1$  there is no constant bound on how many times a node can move. Consider the complete binary tree in Figure 2 and the following execution sequence:

1. We start with an empty set  $S$ .
2. Starting from the root we execute **R1** down the tree, level by level. We do this until we reach a node where we can't execute **R1** anymore or we don't have another level.
3. Starting from the root we execute **R2** down the tree, level by level. We do this until we reach a node where we can't execute **R2** anymore.
4. We repeat Steps 2 and 3 until no more nodes can make a move.

After the above sequence of moves, we obtain a configuration like that shown in Figure 2. The black vertices form the maximal 1-dependent set. In this tree there are four levels numbered 0,1,2, and 3, the root having level 0. All nodes at level  $i$  make exactly  $4 - i$  moves, and 26 moves are made.

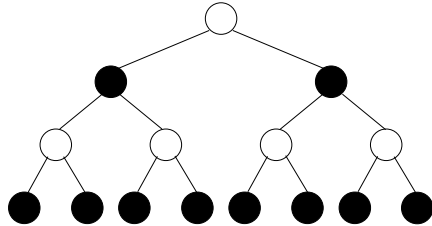


Fig. 2. 1-dependent set in complete binary tree.

This example generalizes as follows. The complete binary tree having  $\ell$  levels numbered  $0, \dots, \ell - 1$  has  $n = 2^\ell - 1$  nodes. As above, an execution can occur in which the  $2^i$  nodes at level  $i$  each make  $\ell - i$  moves. Hence, the running time is

$$\sum_{i=0}^{\ell-1} (\ell - i) 2^i. \tag{2}$$

It is easy to show by induction that (2) equals  $2(2^\ell - 1) - \ell$ , or  $2n - \lceil \log_2 n \rceil$ .

#### 4. Conclusion and related problems

We presented an  $O(kn)$  self-stabilizing algorithm for finding a set which is both maximal  $k$ -dependent and minimal  $(k + 1)$ -dominating. Algorithm 2.1 finds a maximal set of vertices  $S$ , having a characteristic function  $f : V \rightarrow \{0, 1\}$  in which  $f(N(u)) \leq k$  for all  $u \in S$ . We can change the problem by insisting that  $S$  is maximal with respect to the property that the inequality hold for *all* open neighborhoods. That is, we seek a self-stabilizing algorithm for constructing a maximal function  $f : V \rightarrow \{0, 1\}$  such that

$$f(N(u)) \leq k \quad \forall u \in V. \quad (3)$$

We can also consider the *closed neighborhood* of a node  $u$ ,  $N[u] = N(u) \cup \{u\}$ , seeking a maximal  $f$  for which

$$f(N[u]) \leq k \quad \forall u \in V. \quad (4)$$

Finally, we can consider functions  $f : V \rightarrow \{0, 1, \dots, k\}$ , and require either condition (3) or (4).

#### References

- [1] A. Dessmark, K. Jansen, and A. Lingas. The maximum  $k$ -dependent and  $f$ -dependent set problem. In *Algorithms and Computation, 4th International Symposium, ISAAC '93*, pages 88–98. Springer Verlag, 1993. Lecture Notes in Computer Sciences, Vol. 762.
- [2] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17:643–644, 1974.
- [3] K. Diks, O. Garrido, and A. Lingas. Parallel algorithms for finding maximal  $k$ -dependent sets and maximal  $f$ -matchings. *International Journal of Foundations of Computer Science*, 4(2):179–192, 1993.
- [4] H. Djidjev, O. Garrido, C. Levcopoulos, and A. Lingas. On the maximum  $q$ -dependent set problem. In *International Conference for Young Computer Scientists, ICYCS'91*, pages 271–274, 1991.
- [5] O. Favaron. On a conjecture of Fink and Jacobson concerning  $k$ -domination and  $k$ -dependence. *Journal of Combinatorial Theory Ser. B*, 39:101–102, 1985.
- [6] O. Favaron.  $k$ -domination and  $k$ -independence in graphs. *Ars Combinatoria*, 25C:159–167, 1988.
- [7] O. Favaron, S. M. Hedetniemi, S. T. Hedetniemi, and D. F. Rall. On  $k$ -dependent domination. *Discrete Mathematics*, to appear.
- [8] J. F. Fink and M. S. Jacobson.  $n$ -domination in graphs. In Y. Alavi; G. Chartrand; L. Lesniak; D.R. Lick and C.E. Wall, editors, *Graph Theory with Applications to Algorithms and Computer Sciences*, pages 283–300. Wiley Interscience, 1985.
- [9] J. F. Fink and M. S. Jacobson. On  $n$ -domination,  $n$ -dependence and forbidden subgraphs. In Y. Alavi; G. Chartrand; L. Lesniak; D.R. Lick and C.E. Wall, editors, *Graph*

- Theory with Applications to Algorithms and Computer Sciences*, pages 301–311. Wiley Interscience, 1985.
- [10] O. Garrido. *Efficient Parallel Algorithms for Combinatorial Problems*. PhD thesis, Department of Computer Sciences, Lund University, Sweden, 1996.
  - [11] S. M. Hedetniemi, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Self-stabilizing algorithm for minimal dominating and maximal independent sets. Submitted.
  - [12] S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Maximal matching stabilizes in time  $O(m)$ . *Information Processing Letters*, 80:221–223, 2001.
  - [13] Su-Chu Hsu and Shing-Tsaan Huang. A self-stabilizing algorithm for maximal matching. *Information Processing Letters*, 43(2):77–81, 1992.
  - [14] M. S. Jacobson and K. Peters. Complexity questions for n-domination and related parameters. *Congressus Numerantium*, 68:7–22, 1989.
  - [15] G. Tel. Maximal matching stabilizes in quadratic time. *Information Processing Letters*, 49(6):271–272, 1994.
  - [16] G. Tel. *Introduction to distributed algorithms*. Cambridge University Press, second edition, 2000.