Parallel Processing Letters
© World Scientific Publishing Company

DISTANCE-TWO INFORMATION IN SELF-STABILIZING ALGORITHMS

MARTIN GAIRING*
Faculty of Computer Science, Electrical Engineering and Mathematics,
University of Paderborn, 33102 Paderborn, Germany

and

WAYNE GODDARD and STEPHEN T. HEDETNIEMI†
Department of Computer Science,
Clemson University, Clemson, SC 29634, USA
and

PETTER KRISTIANSEN
Department of Informatics,
University of Bergen, N-5020 Bergen, Norway
and

ALICE A. McRAE
Department of Computer Science,
Appalachian State University, Boone, NC 28608, USA

Received (received date)
Revised (revised date)
Communicated by (Name of Editor)

ABSTRACT
In the state-based self-stabilizing algorithmic paradigm for distributed computing, each node has only a local view of the system (seeing its neighbors' states), yet in a finite amount of time the system converges to a global state satisfying some desired property. In this paper we introduce a general mechanism that allows a node to act only on correct distance-two knowledge, provided there are IDs. We then apply the mechanism to graph problems in the areas of coloring and domination. For example, we obtain an algorithm for maximal 2-packing which is guaranteed to stabilize in polynomial moves under a central daemon.

1. Introduction

Self-stabilization is a paradigm for distributed systems that allows a system to achieve a desired, or legitimate, global state, even in the presence of faults. For an overview, see [18, ch. 15] or [3].

*Supported by the IST Program of the EU under contract number IST-2001-33116 (FLAGS).
†Research supported in part by NSF Grant CCR-0222648.

In a common version of self-stabilization, each node executes the same self-stabilizing algorithm, maintains its own set of local variables, and changes the values of its local variables based on the current values of its variables and those of its neighbors. In this paper we introduce a general mechanism that allows a node to act on only correct distance-two knowledge: that is, to act based on the variables of its neighbors' neighbors. In particular, we show how to convert an algorithm which assumes each node can see its neighbors' neighbors to run on the normal model (given that there are IDs): see Theorems 1 and 2.

Our example applications are in the area of graph algorithms. There has been considerable work on self-stabilizing algorithms for graph problems, such as matchings [17,15,16], colorings and independent sets [11,14] and domination [9,14]. One application of the mechanism is to distance-two coloring problems, which generalize for example neighbor-unique numbering [10]. Another application is to domination problems: in particular, we show how the mechanism readily provides a self-stabilizing algorithm for finding a minimal $\{k\}$-dominating function in an arbitrary graph, though this particular problem has also a linear-move anonymous algorithm.

A preliminary version of this paper appeared as [8].

1.1. The algorithmic model

The contents of a node's local variables determine its *local state*. The system's *global state* is the union of all local states. When a node changes its local state, it is said to make a *move*. Our algorithms are given as a set of rules of the form $p(i) \Rightarrow M$, where $p(i)$ is a boolean predicate, and M is a move which describes the changes to be made to the node's local variables. A node i becomes *privileged* if $p(i)$ is true.

When no further moves are possible, we say that the system is *stable* or is in a stable state. (Since our interest is graph algorithms, we focus on so-called *static* or *silent* problems; but these ideas can also be applied to dynamic problems such as distance-two mutual exclusion.) We say that a self-stabilizing algorithm is *correct* if, when the system executes the algorithm, (1) every stable state it can reach is legitimate; that is, every stable state has the desired global property, and (2) it always reaches a stable state after a finite number of moves.

Our main algorithm assumes that each node has a unique ID. We assume there exists a daemon, an adversarial oracle as introduced in [2], which selects one or more of the privileged nodes which then make a move. In the *serial* or *central daemon* model, no two nodes move at the same time (though as usual it is sufficient that adjacent nodes never move at the same time). In the *distributed daemon* model, the daemon can choose any subset of privileged nodes to move simultaneously.

We provide the conversion mechanism first for a central daemon. Though it is known that, given IDs, any algorithm for the central daemon can be transformed into one for the distributed daemon (see for example, [1]), we also show how to adapt the conversion mechanism for a distributed daemon.

While the number of moves is one measure of complexity, another is the notion of *round* [3,4]. A *round* is the minimum period of time where every node that is continually privileged moves at least once.

1.2. Graph notation

A distributed system can be modeled with a connected, undirected graph G with node set $V(G)$ and edge set $E(G)$. If i is a node in $V(G)$, 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. The *closed neighborhood* of node i is the set $N[i] = N(i) \cup \{i\}$. We let $n = |V(G)|$ denote the number of nodes in the graph and $m = |E(G)|$ the number of edges.

2. Distance-Two Knowledge

Consider a model where a node can instantaneously see the states of all nodes within distance two from it. We call this the *extended model*. Certain problems are much easier under this model.

For example, a *2-packing* in a graph is a set S of nodes such that no two nodes in S are adjacent and no two nodes in S have a common neighbor. In the extended model, a self-stabilizing algorithm for the maximal 2-packing problem is straightforward under a central daemon. It is essentially the maximal independent set algorithm. Let $N^2(i)$ denote the set consisting of the nodes within distance two of node i (excluding i itself):

Algorithm 1: 2-Packing assuming extended model

```
ENTER: if $i \notin S \ \wedge \ S \cap N^2(i) = \emptyset$
  then {enter $S$

LEAVE: if $i \in S \ \wedge \ S \cap N^2(i) \neq \emptyset$
  then {leave $S$
```

A node can only enter S once, and hence this algorithm terminates in a linear number of moves.

However, the standard model does not provide knowledge of nodes at distance two. We now provide a mechanism to simulate this knowledge. The solution requires IDs. But some symmetry breaking is necessary. For example, it is easily shown that deterministic maximal 2-packing requires IDs (even on the 6-cycle).

A primitive form of this approach was the self-stabilizing algorithm for maximal 2-packing given in [7].

2.1. Achieving distance-two knowledge

Suppose there is a self-stabilizing algorithm $\mathcal{S}$ that runs on the extended model (where a node can instantaneously see all its neighbors' variables and all the variables at their neighbors). We now show how to implement this algorithm under the normal model.

In our algorithm, we assume that every node has a distinct identifier, or ID, and that there exists a total ordering on the IDs. Each node i has three local variables:

- The variable f stores the state of the node i with respect to the algorithm $\mathcal{S}$. (We use $\mathbb{S}$ to denote the set of all possible states.)

- The variable σ stores a local copy of its neighbors' f-values. Since we have IDs, one may assume that $\sigma(i)$ is, for example, an array indexed by the neighbors' IDs. We say that $\sigma(i)$ is *correct* if it correctly represents the values at the neighbors.

- A pointer stores the ID of a neighbor. We use the notation $i \rightarrow j$ and $i \rightarrow NULL$ to denote that the pointer of i points to j and that the pointer of i points to $NULL$, respectively. A node i can also point to itself, indicated by $i \rightarrow i$.

We say that a node is $\mathcal{S}$-*alive* if it is privileged for $\mathcal{S}$ *assuming that* $\sigma(j)$ *is correct for each neighbor* j. (If the $\sigma(j)$ are contradictory—two neighbors report different information about the same distance-two node—then the node is considered not $\mathcal{S}$-alive.)

For simplicity we define the following quantity:

$$minN[i] = min\{\, j \mid j \in N[i] \wedge j \rightarrow j \,\}, \text{ where } min\{\emptyset\} = NULL.$$

That is, $minN[i]$ is the minimum ID of a node in the closed neighborhood of i that points to itself. If no such node exists, $minN[i] = NULL$.

The pseudocode for the self-stabilizing algorithm is shown in Algorithm 2. The rules ASK and RESET are used to achieve mutual exclusion for the CHANGE moves; and to make sure that when a node makes a CHANGE move, this move is based upon correct σ-values in its neighborhood. It should be noted that a CHANGE move can occur even when a node is not $\mathcal{S}$-alive—a distance-2 neighbor has changed—in order to allow other nodes to progress.

A node i that believes it is $\mathcal{S}$-privileged tries to make an ASK move. It can make this ASK move if it and all its neighbors point to $NULL$. When making an ASK move, node i sets its pointer to itself. The next time that the node (is privileged and) gets scheduled, either it changes its f-value (by a CHANGE move) or it makes a RESET move. The CHANGE move is made if all neighbors of i point to i; when moving, the node updates $f(i)$ (if necessary) and sets its pointer back to $NULL$. The RESET move is made if i has an asking neighbor with ID lower than that of i; this move sets the pointer of i to $minN[i]$, that is, the asking neighbor with smallest ID. In general, the RESET move ensures that the node points to the smallest asking neighbor—Note that a RESET cannot cause a node to point to itself. A node where $\sigma(i)$ is incorrect can UPDATE its $\sigma(i)$.

Algorithm 2: EXECUTING DISTANCE-TWO ALGORITHM $\mathcal{S}$

```
local at node i: $f \in \mathbb{S}$
        $\sigma$, array of $\mathbb{S}$ indexed by neighbors' IDs
        $\rightarrow$, pointer to node

UPDATE-SIGMA: if $\sigma(i)$ incorrect
  then {update $\sigma(i)$

ASK: if $i$ $\mathcal{S}$-alive $\wedge$ $\forall j \in N[i] : j \rightarrow NULL$ $\wedge$ $\sigma(i)$ correct
  then {$i \rightarrow i$

RESET: if $i \not\rightarrow minN[i]$ $\wedge$ $\sigma(i)$ correct
  then {$i \rightarrow minN[i]$

CHANGE: if $\forall j \in N[i] : j \rightarrow i$ $\wedge$ $\sigma(i)$ correct
  then {if $i$ $\mathcal{S}$-alive, then update $f(i)$
        $i \rightarrow NULL$
```

Lemma 1 *If Algorithm 2 terminates, then all pointers are null, $\sigma(i)$ is correct for all i, and no node is $\mathcal{S}$-privileged.*

Proof. Assume that Algorithm 2 has stabilized. Suppose there is a node which points to itself. Then the smallest such node will have all its neighbors pointing to it (else the neighbor would be privileged for RESET). So the asking node will be privileged for CHANGE, a contradiction.

Hence no node points to itself. This implies that $minN[i] = NULL$ for all i, and hence every node points to $NULL$. Since no node is privileged for UPDATE-SIGMA, it follows that all the $\sigma(i)$ are correct. Since no node is privileged for rule ASK, it follows that no node is $\mathcal{S}$-alive and hence no node is $\mathcal{S}$-privileged. $\Box$.

We next show that the algorithm has the desired distance-two property:

Lemma 2 *If a node i makes an ASK move and then later a CHANGE move with no intervening RESET move by i, then the CHANGE move is based upon correct values of $\sigma(j)$ for all neighbors $j \in N(i)$.*

Proof. At the time of the ASK move, all neighbors j have pointer set to $NULL$. At the time of the CHANGE move, all j point to i. At the last point where j changes its pointer back to i, its $\sigma(j)$ was correct. From that point on, since $j \rightarrow i$ no other neighbor of j can make a CHANGE move, and so $\sigma(j)$ remains correct. $\Box$.

We define a REAL-CHANGE move as a CHANGE move where the node changes the value of f.

Lemma 3 *Consider an interval without a REAL-CHANGE move. Then each node i with d_i neighbors can make:*
(a) at most one UPDATE-SIGMA move;

(b) at most one ASK move and at most one CHANGE move; and
(c) at most $O(d_i)$ RESET moves.

Proof. (a) Obvious.

(b) When node i CHANGEs, all neighbors are pointing to it. For it to CHANGE again, it must first ASK. But it can only ASK if all its neighbors are pointing to *NULL*. So all its neighbors must have moved, and their σ-values are correct at the time of the ASK. Thus i is $\mathcal{S}$-alive, and cannot execute another CHANGE without this being a REAL-CHANGE, nor can it execute another ASK.

(c) In between two successive RESET moves, the set of asking neighbors must change. Since a neighbor can enter and leave the set of asking neighbors only once each, it follows that this set can change at most $2d_i$ times. $\Box$.

Lemma 3 shows that the number of moves in between two REAL-CHANGE moves is linear in the number of edges.

Putting this all together we obtain:

Theorem 1 *Consider a network with m edges. If $\mathcal{S}$ is a distance-two knowledge algorithm which converges in A moves on the extended model under the central daemon, then Algorithm 2 implements $\mathcal{S}$ and converges in $O(mA)$ moves under the central daemon.*

Further, we can show that this bound is best possible in that one can construct, for some protocols, networks where the average number of moves between REAL-CHANGE moves is quadratic in n. We omit the construction.

Corollary 1 *Algorithm 1 implemented using Algorithm 2 constructs a maximal 2-packing in guaranteed $O(nm)$ moves.*

2.1.1. Saving resources

It is to be noted that the σ function can be a summary of the neighbors' variables. For example, in implementing the maximal 2-packing algorithm, it is sufficient for $\sigma(i)$ to count how many neighbors are in the set S (and indeed be one of three values: 0, 1 and "many").

2.1.2. Round analysis

The consequences for round-complexity are not as good. The problem is that a node ceases to be privileged when any of its neighbors point away; and so in a network where one node is adjacent to all others, the daemon can ensure that each round has at most one move that represents progress for $\mathcal{S}$. The conversion mechanism does not ensure fairness. Indeed, we can construct a network and protocol where there is convergence in one round in the extended model, but where convergence is not guaranteed at all after conversion. We omit the construction.

It is trivial that the number of rounds cannot exceed the number of moves. But we can slightly improve on the bound.

Lemma 4 *Consider a network with n nodes. If $\mathcal{S}$ is a distance-two knowledge algorithm which converges in A moves on the extended model under the central daemon, then Algorithm 2 implements $\mathcal{S}$ and converges in $O(nA)$ rounds under the central daemon.*

We omit the proof.

2.2. Distributed daemon

So far we assumed that the daemon can only choose one of the privileged nodes to move. We will now describe how Algorithm 2 can be extended to work under a *distributed daemon.* Indeed, the mechanism provides local mutual exclusion, so the results hold even if the original algorithm $\mathcal{S}$ was only for a central daemon.

The only problem for correctness under the distributed daemon, is that Lemma 2 does not hold. For, it can happen that some node j makes an RESET move to point to i at the same time a neighbor k of j makes a CHANGE move making $\sigma(j)$ incorrect.

The solution is to use a *dirty bit* at every node: this is set whenever a node changes its pointer (from one asking node to another such node). We change the rules as follows:

1. We add a rule CLEAR-BIT which resets the dirty bit.
2. The predicate for each of the original rules is adjusted so that none can fire when the dirty bit is set.
3. The predicate for CHANGE is further adjusted so that a node cannot make a CHANGE move when any of its neighbors has its dirty bit set.

Theorem 2 *Consider a network with n nodes and m edges. If $\mathcal{S}$ is a distance-two knowledge algorithm which converges in A moves on the extended model under the central daemon, then Algorithm 2 as adjusted above implements $\mathcal{S}$ and converges in $O(n^2mA)$ steps under the distributed daemon.*

Proof. By the same proof as in Lemma 3, while there is no REAL-CHANGE move, each node can execute UPDATE-SIGMA at most once and CHANGE at most once. Thus there are at most n CHANGE moves in between two consecutive REAL-CHANGE moves. So consider an interval where there is no CHANGE move.

A group of nodes can ASK simultaneously, but the smallest ID in this group will not retract and it freezes its neighbors. Thus there can be at most n steps where some node ASKs.

In particular, a node enters or leaves the set S of self-pointing nodes at most n times. Each RESET move by node i is in response to a change in $S \cap N(i)$. Hence, it can execute RESET at most $O(nd_i)$ times, where d_i is its degree.

Hence there are at most $O(nm)$ steps between CHANGE moves, and therefore at most $O(n^2m)$ steps in between REAL-CHANGE moves. The theorem follows. $\Box$.

Further, we can show that this bound is best possible in that one can construct, for some protocols, networks where the average number of moves between REAL-CHANGE moves is $\Theta(n^2m)$. We omit the construction.

2.3. Coloring

We show how to apply the above mechanism to provide the first self-stabilizing algorithm for coloring with distance-two constraints.

Consider a coloring problem where a node must be colored such that there is some restriction on its color given the colors of its neighbors and their neighbors. Specifically, there is disallowed set D_0 of unordered pairs of colors for adjacent nodes and a disallowed set D_1 of unordered pairs of colors for nodes at distance 2. For example, an $L(2,1)$-coloring [12] is a coloring with the integers such that adjacent nodes have colors whose difference is at least two and distance-two nodes have distinct colors. Suppose further that each node i has a list $L(i)$ of possible colors $L(i)$.

Then the greedy algorithm is naturally expressed by the following idea. The value $f(i)$ will be the color of node i. Let $D(i)$ be the disallowed colors for node i (based on D_0, D_1 and the colors of i's neighbors and their neighbors): it is assumed that $L(i)$ is bigger than $D(i)$ can ever be.

Algorithm 3: Coloring assuming extended model

```
RECOLOR: if $f(i) \in D(i)$
  then {set $f(i)$ to be any element of $L(i) - D(i)$
```

Lemma 5 *The coloring algorithm in the extended model converges in at most n moves.*

Proof. Since we assumed that the restrictions are unordered pairs, when a node moves it does not create any new problems for its neighbors. $\Box$.

Together with Theorem 1, this shows that the self-stabilizing coloring algorithm stabilizes in the normal model.

3. {k}-Domination

A *dominating set* is a set $S \subseteq V(G)$ such that $N[i] \cap S \neq \emptyset$ for all $i \in V(G)$. There are many generalizations. For a function $f\colon V(G) \to \{0,1,\ldots,k\}$, let $f(S) = \sum_{v\in S} f(v)$ for a subset $S \subseteq V(G)$. Such a function is a $\{k\}$*-dominating function* [5,6,13], if for every $i \in V(G)$ we have $f(N[i]) \geq k$. The case $k = 1$ is a normal dominating set. A $\{k\}$-dominating function f is *minimal* if there does not exist a $\{k\}$-dominating function f' with $f' \neq f$ and $f'(i) \leq f(i)$, $\forall i \in V(G)$. Equivalently, for every node i, if $f(i) > 0$, then there is a node $j \in N[i]$ with $f(N[j]) = k$.

The general idea of $\{k\}$-domination is that every node in the network requires a specific amount of some resource, for example k units. It then becomes a resource allocation problem to distribute the least amount of the resource among the nodes so that each node can access at least k units within its closed neighborhood—the node must either supply itself or use the resources of its neighbors. Examples of this resource might include a minimum number of firefighters, ambulances, or emergency vehicles, or a minimum number of CPU cycles, memory or servers.

Of course, in the most general setting, nodes have different resource needs. Thus, we can associate with each node i a minimum resource requirement r_i. We then seek an allocation of resources, at minimum cost, such that the total resources assigned to each closed neighborhood $N[i]$ is at least r_i. In the definition of $\{k\}$-domination it is assumed that all nodes have the same resource requirement, viz. k, but the algorithms that follow can easily be adapted.

3.1. Self-stabilizing minimal $\{k\}$-dominating function

It is to be noted that [9] presents a self-stabilizing algorithm for finding a minimal total dominating set in an arbitrary graph and generalizes this to other dominating functions. While this generalization provides a self-stabilizing algorithm for $\{k\}$-domination, it involves the use of k pointers, and is more complicated than the algorithms presented here.

In the extended model there is a straight-forward algorithm. A node increases its value if it has insufficient resources, and lowers its value if it and all its neighbors have excess resources. If S is a set of nodes, we use the notation $f(S) = \sum_{v \in S} f(v)$.

Algorithm 4: $\{k\}$-DOMINATION ASSUMING EXTENDED MODEL

```
INCREASE: if $f(N[i]) < k$
  then $\{f(i) = k - f(N(i))$

DECREASE: if $f(i) > 0 \ \wedge \ \forall j \in N[i] : f(N[j]) > k$
  then $\{f(i) = max\{\, 0,\, k + f(i) - min_{j \in N[i]} f(N[j])\,\}$
```

Lemma 6 *For Algorithm 4 executed in the extended model, each node will move at most k+1 times, and the result will be a minimal $\{k\}$-dominating function.*

Proof. The first move by node i ensures that $f(N[i]) \geq k$. Thus it will execute INCREASE at most once. It can then decrease at most k times (its maximum value is k and its minimum value is 0).

At termination, the function is $\{k\}$-dominating since no node needs to INCREASE, and minimal since no node can DECREASE. □.

Corollary 2 *Algorithm 4 implemented using Algorithm 2 constructs a minimal $\{k\}$-dominating function in guaranteed $O(knm)$ moves under the central daemon.*

3.2. Minimal $\{k\}$-domination in linear time

Though this problem was the original application for the paradigm, it turns out that distance-two knowledge is actually unnecessary for $\{k\}$-domination. It is sufficient that a node worry only about itself.

Define for node i the *threshold* function

$$t(i) = max(\, k - f(N(i)),\, 0)\,.$$

Then an assignment f of integers is $\{k\}$-dominating iff $f(i) \geq t(i)$ for all i. If f is not minimal—meaning the assignment of some node i can be lowered—then $f(i) > t(i)$. Hence:

Lemma 7 *If $f(i) = t(i)$ for all i, then f is minimal $\{k\}$-dominating.*

We define a self-stabilizing algorithm with the single rule.

Algorithm 5: Linear-time $\{k\}$-domination

```
local at node i: $f(\_) \in \{0, 1, \ldots, k\}$
MOVE: if $f(i) \neq t(i)$
  then $\{f(i) = t(i)$
```

The above lemma shows that if this algorithm stabilizes, then the result is a minimal $\{k\}$-dominating function. We next show that the algorithm stabilizes.

Theorem 3 *Algorithm 5 stabilizes with a $\{k\}$-dominating function in at most $2k^2m + O(n)$ moves.*

Proof. Define the two sums

$$V = \sum_i f(i)\,(2k - f(i)) \quad \text{and} \quad W = 2\sum_{i\sim j} f(i)f(j) = \sum_i f(i)f(N(i)).$$

Consider a potential function $\phi = W - V$. Since $\phi \geq -V \geq -nk^2$ and $\phi \leq W \leq 2k^2m$, the result follows from the following lemma. □.

Lemma 8 *Each move decreases ϕ.*

Proof. Consider first a move that increases the weight of some node: say node i changes from $f(i) = x$ to $f(i) = y$. Then it follows that $f(N(i)) = k - y$. Thus the increase in V is $y(2k - y) - x(2k - x)$ and the increase in W is $2(y - x)(k - y)$. Hence,

$$\Delta\phi = 2(y - x)(k - y) - (y(2k - y) - x(2k - x)) = -(y - x)^2 < 0.$$

A move that decreases $f(i)$ from y to x is similar, except that in the case $x = 0$ all we can say is that the decrease in W is at least $2yk$. Thus

$$\Delta\phi \leq 2(x - y)(k - x) - (x(2k - x) - y(2k - y)) = -(y - x)^2 < 0.$$

This concludes the proof of the lemma and the theorem. □.

If each node has its own target $k(i)$, then the potential function is changed by making $V = \sum_i f(i)(2k(i) - f(i))$, and the proof of stabilization is the same.

3.2.1. Limits on the algorithm

It should be noted that the converse of Lemma 7 does not hold. It follows that Algorithm 5 is not stable for all minimal $\{k\}$-dominating functions. Algorithm 4, on the other hand, can produce any minimal $\{k\}$-dominating function.

3.2.2. Improving the bound

One can improve the constant factor in the running time. Define an edge as *bad* if the sum of the weights of its endpoints exceeds k; it is good otherwise. It follows that for a good edge $\{i, j\}$ it holds that $f(i)f(j) \leq k^2/4$. It is clear that bad edges cannot be created.

Then let W' be the sum as in W but taken only over good edges, and let $\phi' = W' - V$. We omit the proof of the following lemma.

Lemma 9 *Each move either decreases ϕ'; or it decreases the number of nodes involved in bad edges while increasing ϕ' by at most $bk^2/2$ where b is the number of bad edges that become good.*

The total increase in ϕ' over all bad–good conversions is at most $k^2/2$ times the total number of original bad edges. The starting value of ϕ' is at most $k^2/2$ times the total number of original good edges. Since there are at most n moves that turn bad edges into good, it follows that the number of moves is at most

$$k^2m/2 + O(k^2n).$$

For $k = 2$, this gives $2m + O(n)$. By a more careful analysis, and a small modification to the algorithm, this can be made as low as $2m + 3n$. (See the $\{2\}$-domination algorithm in the preliminary version [8].)

3.3. Complexity

Surprisingly perhaps, $\{k\}$-domination has not been established as intractable. The concept of a $\{k\}$-dominating function gives rise to the following decision problem:

$\{k\}$-DOMINATING FUNCTION
INSTANCE: A graph G and a positive integer l.
QUESTION: Does G have a $\{k\}$-dominating function f, with $f(V(G)) \leq l$?

Theorem 4 $\{k\}$-DOMINATING FUNCTION *is $\mathcal{NP}$-complete for all $k \geq 1$.*

Since the focus of this paper is on self-stabilization, we refer the reader to the preliminary version [8] for the proof.

4. Concluding Remarks

In this paper we considered an extended model where a node can see the variables of the nodes at distance 2. We presented a general conversion that allows one to emulate an algorithm for the extended model on the normal one (assuming IDs) with a polynomial slow down. We showed that this paradigm could be applied to problems in coloring and domination.

This approach can certainly be extended to a model where a node can see the variables of farther nodes. It can also be extended to handle dynamic problems, such as local mutual exclusion, though it does not ensure fairness.

Acknowledgments

The authors are very grateful to the referees for making several suggestions which greatly improved the results in this paper. In particular, one referee of the preliminary version [8] suggested a special case of the linear-time $\{k\}$-domination algorithm, and one referee of the final version pointed out an improvement in the conversion algorithm that made it polynomial.

[1] J. Beauquier, A. K. Datta, M. Gradinariu, and F. Magniette. Self-stabilizing local mutual exclusion and daemon refinement. In *DISC00 Distributed Computing 14th International Symposium, Springer LNCS:1914*, pages 223–237, 2000.

[2] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17:643–644, 1974.

[3] S. Dolev. *Self-Stabilization.* MIT Press, 2000.

[4] S. Dolev, A. Israeli, and S. Moran. Uniform Dynamic Self-Stabilizing Leader Election. *IEEE Trans. on Parallel and Distributed Systems*, volume 8, no. 4, 1995.

[5] G. S. Domke. *Variations of Colorings, Coverings, and Packings of Graphs.* PhD thesis, Clemson University, 1988.

[6] G. S. Domke, S. T. Hedetniemi, R. C. Laskar, and G. Fricke. Relationships between integer and fractional parameters of graphs. In Y. Alavi et al. eds, *Graph Theory, Combinatorics, and Applications, Proceedings of the Sixth Quadrennial Conference on the Theory and Applications of Graphs (Kalamazoo MI, 1988)*, volume 2, pages 371–387. Wiley, 1991.

[7] M. Gairing, R. M. Geist, S. T. Hedetniemi, and P. Kristiansen. A self-stabilizing algorithm for maximal 2-packing. *Nordic Journal of Computing*, 11(1):1–11, 2004.

[8] M. Gairing, S. T. Hedetniemi, P. Kristiansen, and A. A. McRae. A self-stabilizing algorithm for {k}-domination. In *SSS03 Sixth International Symposium on Self-Stabilizing Systems, Springer LNCS 2704*, pages 49–60, 2003

[9] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. A self-stabilizing distributed algorithm for minimal total domination in an arbitrary system graph. In *Proceedings of the 8th International Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA'03)*, 2003.

[10] M. Gradinariu and C. Johnen. Self-stabilizing neighborhood unique naming under unfair scheduler. In *Proceedings of Euro-Par'01 Parallel Processing*, LNCS:2150, Springer, 2001, 458–465.

[11] M. Gradinariu and S. Tixeuil. Self-stabilizing vertex coloration of arbitrary graphs. In *4th International Conference On Principles Of Distributed Systems, OPODIS'2000*, Studia Informatica Universalis, 2000, 55–70.

[12] J. R. Griggs and R. K. Yeh. Labelling graphs with a condition at distance 2. *SIAM J. Discrete Math.* 5(4):586–595, 1992.

[13] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Fundamentals of Domination in Graphs.* Marcel Dekker, 1998.

[14] S. M. Hedetniemi, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Self-stabilizing algorithm for minimal dominating and maximal independent sets. *Computers & Mathematics with Applications*, 46(5–6):805–811, 2003.

[15] 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.

[16] Su-Chu Hsu and Shing-Tsaan Huang. A self-stabilizing algorithm for maximal matching. *Information Processing Letters*, 43(2):77–81, 1992.

[17] G. Tel. Maximal matching stabilizes in quadratic time. *Information Processing Letters*, 49(6):271–272, 1994.

[18] G. Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, second edition, 2000.