Distance-k Information in Self-stabilizing Algorithms*

Wayne Goddard[1], Stephen T. Hedetniemi[1], David P. Jacobs[1], and Vilmar Trevisan[2]

[1] Department of Computer Science, Clemson University, SC 29634 USA
{goddard, hedet, dpj}@cs.clemson.edu
[2] Instituto de Matemática, UFRGS, Porto Alegre, Brazil
trevisan@mat.ufrgs.br

Abstract. Many graph problems seem to require knowledge that extends beyond the immediate neighbors of a node. The usual self-stabilizing model only allows for nodes to make decisions based on the states of their immediate neighbors. We provide a general polynomial transformation for constructing self-stabilizing algorithms which utilize distance-k knowledge, with a slowdown of $n^{O(\log k)}$. Our main application is a polynomial-time self-stabilizing algorithm for finding maximal irredundant sets, a problem which seems to require distance-4 information. We also show how to find maximal k-packings in polynomial-time. Our techniques extend results in a recent paper by Gairing et al. for achieving distance-two information.

1 Introduction

Self-stabilization, introduced by Dijkstra [1], is the most inclusive approach to fault tolerance in distributed systems. In a self-stabilizing algorithm, each node maintains its local variables, and can make decisions based on the correct knowledge of its neighbors' states. In a self-stabilizing algorithm, a node may change its local state by making a *move* (an action which causes a change of local state). 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. 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, we cannot predict which node will move next. In this paper we say that an algorithm *stabilizes* if no node is privileged. An execution will be represented as a sequence of moves $M_1, M_2, \ldots$, in which M_s denotes the s-th move. One can transform the algorithm to work under other daemons, using established techniques. We refer the reader to [2] for a general treatment of self-stabilizing algorithms.

* Research supported by: NSF grant CCR-0222648; CNPq grant 453991/2005-0; and FAPERGS grant 05/2024.1.

A distributed system can be modeled with an undirected graph $G = (V, E)$, where V is a set of n nodes and E is a set of m edges. If $i \in V$, then $N(i)$, its *open neighborhood*, denotes the set of nodes to which i is adjacent, and $N[i] = N(i) \cup \{i\}$ denotes its *closed neighborhood*. Every node $j \in N(i)$ is called a *neighbor* of node i. Throughout this paper we assume G is connected and $n > 1$.

In the usual self-stabilizing model, each node i can read only the variables of its neighbors, that is, those nodes which are a distance of one from i. In this paper, we show how to obtain self-stabilizing algorithms in which a node i can effectively read the contents of variables which are within distance k of i, for any fixed $k \geq 1$, extending results in [3] for achieving distance-two information. This will result in a slowdown of $n^{O(\log k)}$. In Section 3, we obtain a polynomial time self-stabilizing algorithm for finding a maximal irredundant set, a problem which requires distance-4 information.

We assume throughout this paper that all nodes have a unique integer ID. Sometimes we do not distinguish between a node i and its ID. For each $k \geq 1$, we let $N^k[i]$ denote the set of nodes whose distance from i is at most k, and we let $N^k(i) = N^k[i] - \{i\}$. When $k = 1$, these sets correspond, respectively, to the closed and open neighborhoods of i.

A *k-packing* in a graph $G = (V, E)$ is a set $S \subseteq V$ of nodes such that for every pair of distinct nodes, $u, v \in S$, their minimum distance $d(u, v) > k$. A 1-packing is, therefore, a set S having the property that no two nodes in S are adjacent ($d(u, v) > 1$). This is normally called an *independent* set.

Algorithm 1.1 is a well-known and simple self-stabilizing algorithm for finding the characteristic function of a maximal independent set. It is easy to show that this algorithm stabilizes in at most $2n$ moves [6] in the distance-1 model.

Algorithm 1.1. MAXIMAL INDEPENDENT SET

```
local variable: $f$
ENTER: if $f(i) = 0 \wedge (\forall j \in N(i))(f(j) = 0)$
  then $f(i) = 1$
LEAVE: if $f(i) = 1 \wedge (\exists j \in N(i))(f(j) = 1)$
  then $f(i) = 0$
```

2 Distance-*k* problems

In [3], it was observed that certain algorithmic problems can be solved more easily on an *extended model* in which each node can instantly see all state information of nodes that are within distance *two*. Having done this, the extended model can be simulated using a conventional self-stabilizing algorithm, provided all nodes have unique IDs. In this paper we show how arbitrary distances greater than two can be achieved. Our idea is to use the technique in [3] recursively.

We now define a class of self-stabilizing algorithm models. For each $k \geq 1$, in the *distance-k self-stabilizing model*, each node i can instantly see all state information of all nodes in $N^k[i]$. We assume that node i can read the ID of j and its state information $f(j)$ for each $j \in N^k[i]$. For brevity, we refer to this as the *distance-k model.* The distance-1 model is the usual self-stabilizing algorithmic model. It will be convenient to assume for now that k is a power of two.

Now let $k = 4$, and consider Algorithm 2.1, which assumes the distance-4 model. If Algorithm 2.1 stabilizes, the set $S = \{i \mid f(i) = 1\}$ is a maximal 4-packing. For if no node is privileged to LEAVE, then S must be a 4-packing, and if no node can ENTER, the 4-packing is maximal. Moreover, the algorithm must always stabilize. Indeed, once a node makes an ENTER move, no node in $N^4(i)$ can ENTER, and so no node in $N^4[i]$ can move again. If a node makes a LEAVE move, its next move must be an ENTER, after which it cannot move. It follows that

Lemma 1. *The distance-4 Algorithm 2.1 finds a maximal 4-packing in at most* $2n$ *moves.*

Algorithm 2.1. MAXIMAL 4-PACKING IN DISTANCE 4

```
local variable: $f$
ENTER: if $f(i) = 0 \wedge (\forall j \in N^4(i))(f(j) = 0)$
  then $f(i) = 1$
LEAVE: if $f(i) = 1 \wedge (\exists j \in N^4(i))(f(j) = 1)$
  then $f(i) = 0$
```

Assume now that we have some distance-$2k$ algorithm $\mathcal{S}_{2k}$, such as Algorithm 2.1, in which every node has a local variable f. We now will describe a way to simulate $\mathcal{S}_{2k}$ using a distance-k algorithm $\mathcal{S}_k$. We will see that the running times of $\mathcal{S}_k$ and $\mathcal{S}_{2k}$ are related to within a factor in $O(n^3)$. In Algorithm $\mathcal{S}_k$, each node i has three local variables:

- The variable f stores the state of node i with respect to $\mathcal{S}_{2k}$, that is, the value of $\mathcal{S}_{2k}$'s local variable.
- The variable σ stores a local copy of $f(j)$ for each $j \in N^k(i)$. We may assume that $\sigma(i)$ is a list of pairs of the form (j, f_j), where j is an ID of a node in $N^k(i)$. We say that $\sigma(i)$ is *correct* if for all $j \in N^k(i)$, $f(j) = f_j$.
- A pointer stores the ID of a member of $N^k[i]$, or has the value NULL. We write $i \to j$, $i \to i$, and $i \to$ NULL to mean, respectively, that i points to j, i points to itself, and i's pointer is NULL.

At each step in the execution of $\mathcal{S}_k$, the values $f(i)$ represent a state with respect to $\mathcal{S}_{2k}$. A node i in the distance-k model can read directly only state information of nodes in $N^k(i)$. However if $j' \in N^{2k}(i)$, then $j' \in N^k(j)$ for some $j \in N^k[i]$. It follows that in the distance-k model, by reading $\sigma(j)$, node i has a *view* of $f(j')$. However, it is possible for this view to be incorrect.

During the execution of $\mathcal{S}_k$, we say that node i is $\mathcal{S}_{2k}$-*alive* if it is privileged for $\mathcal{S}_{2k}$, under the assumption that its view of $\{(j, f(j)) \mid j \in N^{2k}(i)\}$ is correct.

We define

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

That is, $minN^k[i]$ is the smallest ID, within distance k of i, which is pointing to itself; $minN^k[i]$ is defined to be NULL if no member of $N^k[i]$ points to itself.

Algorithm $\mathcal{S}_k$ is displayed as Algorithm 2.2. When $k = 1$, it is exactly the algorithm described in [3].

Algorithm 2.2. $\mathcal{S}_k$

```
comment: Simulates distance-$2k$ algorithm $\mathcal{S}_{2k}$
local variables: $f, \sigma, \rightarrow$
UPDATE-$\sigma$: if $\sigma(i)$ is incorrect
  then update $\sigma(i)$
ASK: if $i$ is $\mathcal{S}_{2k}$-alive $\wedge$ $(\forall j \in N^k[i] : j \rightarrow$ NULL$)$ $\wedge$ $\sigma(i)$ is correct
  then $i \rightarrow i$
RESET: if $i \not\rightarrow minN^k[i] \wedge \sigma(i)$ is correct
  then $i \rightarrow minN^k[i]$
CHANGE: if $\forall j \in N^k[i] : j \rightarrow i \wedge \sigma(i)$ is correct
  then { if $i$ is $\mathcal{S}_{2k}$-alive, then update $f(i)$
       { $i \rightarrow$ NULL
```

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

Proof. Assume the algorithm has stabilized. Then no node points to itself, for otherwise the node i pointing to itself having the smallest ID would have all members of $N^k[i]$ pointing to it, and i would be privileged for a CHANGE move. Since no node points to itself, $minN^k[i]$ is *NULL*, and therefore all pointers are *NULL*. All $\sigma(i)$ are correct since no node is privileged for an UPDATE-σ. No node is $\mathcal{S}_{2k}$-privileged, for otherwise it would be privileged to execute ASK.

Lemma 3. *While i is pointing to itself, no node in $N^k(i)$ can execute an ASK or CHANGE.*

Proof. For $j \in N^k(i)$ to execute ASK, i must be NULL. For j to execute CHANGE, i must be pointing to j.

Lemma 4. *If i makes an ASK move, its next move must be a CHANGE move.*

Proof. When i makes an ASK move, all members of $N^k[i]$ are NULL. Suppose its next move is a RESET. Then this means that some $j \in N^k(i)$ is pointing to itself. But this is impossible because $i \rightarrow i$. Nor can its next move be an UPDATE-σ, because at the time of the ASK move, $\sigma(i)$ was correct. But this can't change by Lemma 3, nor can its next move be another ASK move because $i \rightarrow i$.

Let us say that a move by i is *correct* if $\sigma(j)$ is correct for all $j \in N^k(i)$, and *incorrect* otherwise.

Lemma 5. *If node i makes an ASK move, then its next CHANGE move is correct.*

Proof. Let j be some member of $N^k[i]$. During the interval between the ASK and CHANGE moves, j must have changed its pointer from NULL to i, at which time $\sigma(j)$ was correct, and there must have been a last time during this interval when this occurred. But $\sigma(j)$ must have remained correct, because no member of $N^k[j]$ could have performed a CHANGE while j was pointing to i.

Lemma 6. *If a node i makes a CHANGE move, then its next ASK move is correct.*

Proof. During this interval, all $j \in N^k[i]$ changed their pointers from i to NULL. There is a last time at which j became NULL, prior to the ASK move. At this time, $\sigma(j)$ is correct, and it must remain so up until the ASK move, since no member of $N^k[j]$ could have performed a CHANGE move as long as j's pointer is NULL.

Lemma 7. *Between any two RESET moves made by i, some some $j \in N^k[i]$ must execute an ASK or a CHANGE.*

Proof. This is clear.

For convenience, we define a REAL-CHANGE move as a CHANGE move in which the variable f is assigned. We let $d_i^k = | N^k(i) |$.

Lemma 8. *Consider an interval without a REAL-CHANGE move. Then each node i can make:*

1. *at most one UPDATE-σ move;*
2. *at most one ASK move;*
3. *at most one CHANGE move; and*
4. *$O(d_i^k)$ RESET moves.*

Proof. 8.1 is obvious. To see 8.2, suppose i makes an ASK move. By Lemma 4, its next move must be a CHANGE move. Then by Lemma 5, the CHANGE move is correct. Since this is not a REAL-CHANGE, i is not $\mathcal{S}_{2k}$-privileged. Since no other REAL-CHANGE moves occur, i cannot become $\mathcal{S}_{2k}$-alive again to execute another ASK move. To see 8.3, suppose i makes a CHANGE move, and then makes an ASK move. By Lemma 6, the ASK move is correct. Since no REAL-CHANGE can take place, the σ's remain the same, and if i were to execute another CHANGE move, it would have to be a REAL-CHANGE. Finally, 8.4 follows from Lemma 7.

Lemma 9. *There are at most $O(n^2)$ moves during an interval without REAL-CHANGE moves.*

Proof. This follows immediately from Lemma 8.

Lemma 10. *Each node can make at most one incorrect REAL-CHANGE move.*

Proof. An incorrect REAL-CHANGE move can only occur as a node's first CHANGE move, because subsequent CHANGE moves will be preceded by an ASK move, which by Lemma 5, must be correct.

Lemma 11. *Let (M_i) be a sequence of moves made by Algorithm 2.2 during which no incorrect REAL-CHANGE occurs. Then the subsequence (M'_i) of REAL-CHANGE moves is a valid computation of $\mathcal{S}_{2k}$.*

Proof. This is clear.

Lemma 12. *Suppose Algorithm $\mathcal{S}_{2k}$ can execute at most A moves. Then in any interval without an incorrect REAL-CHANGE move, Algorithm $\mathcal{S}_k$ can execute at most $O(An^2)$ moves.*

Proof. By Lemma 11, there can be at most A REAL-CHANGE moves, and by Lemma 9, between any two REAL-CHANGE moves, there are at most $O(n^2)$ moves.

Theorem 1. *In a network with n nodes, a distance-$2k$ algorithm $\mathcal{S}_{2k}$ that stabilizes within A moves can be implemented with a distance-k algorithm $\mathcal{S}_k$ that stabilizes in $O(An^3)$ moves.*

Proof. By Lemma 10 there can be at most n incorrect REAL-CHANGE moves. By Lemma 12, during the intervals without incorrect moves, there can be at most $O(An^2)$ moves. Finally by Lemma 2, the algorithm is correct.

By repeating Theorem 1, we obtain

Theorem 2. *In a network with n nodes, a distance-k algorithm $\mathcal{S}_k$ which stabilizes in A moves can be implemented in the distance-1 model by an algorithm that stabilizes in $O(An^{3\lceil \log_2(k) \rceil})$ moves.*

Corollary 1. *There is a self-stabilizing algorithm to find a maximal 4-packing that stabilizes in $O(n^7)$moves.*

Proof. This follows by Lemma 1 and Theorem 2.

When we translate, say, a distance-4 algorithm $\mathcal{S}_4$ to a distance-2 algorithm $\mathcal{S}_2$, each node will contain the original variable f used in $\mathcal{S}_4$ in addition to a pointer and a σ. Note that when $\mathcal{S}_2$ is then translated to a distance-1 algorithm $\mathcal{S}_1$, each node will contain these three variables in addition to another pointer and another σ.

A maximal 4-packing can be found by using a single boolean variable in the distance-4 model. However, to find a maximal 3-packing in the distance-4 model, nodes must know the distances of their neighbors. If we assume that in addition to its usual variables, each node displays a list of the IDs of its neighbors, then in the distance-1 model, each node can compute the subgraph induced by its closed neighborhood. In the distance-k model, each node i can compute the subgraph induced by $N^k[i]$, and can compute, for example, the distance $d(i,j)$ for $j \in N^k[i]$. This is illustrated in Algorithm 2.3. This generalizes to a polynomial time self-stabilizing algorithm for maximal k-packing, for any fixed k, and improves upon the maximal k-packing algorithm in [4] that was given without analysis.

Algorithm 2.3. Distance-4 algorithm for maximal 3-packing

```
local variable: $f$
ENTER: if $f(i) = 0 \wedge (\forall j \in N^4(i), d(i,j) \leq 3)(f(j) = 0)$
  then $f(i) = 1$
LEAVE: if $f(i) = 1 \wedge (\exists j \in N^4(i), d(i,j) \leq 3)(f(j) = 1)$
  then $f(i) = 0$
```

3 Maximal Irredundant Sets

Given a set S of nodes, we say a node $s \in S$ has a *private neighbor* with respect to S if there exists some $x \in N[s] - N[S - \{s\}]$. A set S is *irredundant* [5] if every $s \in S$ has a private neighbor with respect to S. Self-stabilizing algorithms have been found for many kinds of related sets, such as maximal independent sets and minimal dominating sets [6], but finding maximal irredundant sets has proven difficult because the problem seems to require distance-4 knowledge.

Let S be a set of nodes, not necessarily irredundant, and let $s \in S$. If s has a private neighbor with respect to S, but s has no private neighbor with respect to $S \cup \{x\}$, we say x *destroys* s. Finally, we say $x \in V - S$ is *safe* if x has a private neighbor with respect to $S \cup \{x\}$, and no $s \in S$ is destroyed by x.

Consider Algorithm 3.1. It is easy to see that if this algorithm stabilizes, then $S = \{i \mid f(i) = 1\}$ is maximal irredundant. For if it is not irredundant, some i is privileged to execute a LEAVE move. And if it is not maximal irredundant, some i can execute an ENTER move. Note also that once a node executes an ENTER, it will never execute a LEAVE. Thus, given a sufficiently powerful model, each node moves at most twice.

Algorithm 3.1. Maximal irredundant set

```
local variable: $f$
ENTER: if $f(i) = 0 \wedge i$ is safe
  then $f(i) = 1$
LEAVE: if $f(i) = 1 \wedge i$ has no private neighbor
  then $f(i) = 0$
```

Lemma 13. *Node i can decide if it has a private neighbor from the information in $N^2[i]$.*

Proof. A node x is a private neighbor of i if and only if $x \in N[i]$, but for all $j \in N^2(i)$, $j \in S$ implies $x \notin N[j]$.

Lemma 14. *Node i can decide if it is safe from the information in $N^4[i]$.*

Proof. If node i is not safe, then it must destroy some node $j \in N^2[i]$. However, to know whether such a node j has a private neighbor requires examining $\{f(j') \mid j' \in N^2[j]\}$.

Theorem 3. *There is a self-stabilizing algorithm for finding a maximal irredundant set that stabilizes in $O(n^7)$ moves.*

Proof. By Lemma 13 and Lemma 14 it follows that Algorithm 3.1 can be implemented in the distance-4 model. By our earlier comments, Algorithm 3.1 stabilizes in a linear number of moves. The analysis follows by Theorem 2.

We observe that while Algorithm 3.1 makes a linear number of moves in the distance-4 model, each simulated move may not take constant time, although it will be polynomial.

References

1. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Comm. ACM **17** (11) (1974) 643–644
2. Dolev, S.: *Self-Stabilization.* MIT Press, 2000
3. Gairing, M., Goddard, W., Hedetniemi, S.T., Kristiansen, P., McRae, A.A.: Distance-two information in self-stabilizing algorithms, Parallel Process. Lett., **14** (2004) 387–398
4. Goddard, W., Hedetniemi, S.T., Jacobs, D.P., Srimani, P.K.: Self-stabilizing global optimization algorithms for large network graphs, Int. J. Dist. Sensor Net., **1** (2005) 329–344
5. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: *Fundamentals of Domination in Graphs*, Marcel Dekker, New York, 1998
6. Hedetniemi, S.M., Hedetniemi, S.T, Jacobs, D.P., Srimani, P.K.: Self-stabilizing algorithms for minimal dominating sets and maximal independent sets, Comput. Math. Appl., **46** (2003) 805–811