

Distance- k knowledge in self-stabilizing algorithms [★]

Wayne Goddard¹, Stephen T. Hedetniemi¹,
David P. Jacobs¹, and Vilmar Trevisan²

¹ School of Computing, 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 transformation for constructing self-stabilizing algorithms which utilize distance- k knowledge. Our transformation has both a slowdown and space overhead in $n^{O(\log k)}$, and might be thought of as a distance- k resource allocation algorithm. Our main application is a polynomial-time self-stabilizing algorithm for finding maximal irredundant sets, a problem which seems to require distance-4 information. These results can be generalized to efficiently find maximal \mathcal{P} -sets, for properties \mathcal{P} which we call *local monotonic*. Our techniques extend results in a recent paper by Gairing et al. for achieving distance-two information.

1 Introduction

Self-stabilization, introduced by Dijkstra [5], 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*. 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, \dots , in which M_s

[★] This paper is an expanded version of [9], presented at SIROCCO06. This research was supported by: NSF grant CCR-0222648; CNPq grant 453991/2005-0; and FAPERGS grant 05/2024.1.

denotes the s -th move. One can transform the algorithm to work under other daemons, using established techniques. We refer the reader to [6] for a general treatment of self-stabilizing algorithms.

In the usual self-stabilizing model, each node i can read only the variables of its neighbors, that is, those nodes which are within distance one from i . In the next section we introduce a more general model in which nodes can read within distance k . In Section 3, we show how distance- k information, for any fixed $k > 0$, can be achieved in the self-stabilizing model, extending results in [7] when $k = 2$. This will result in a slowdown, where the running time is multiplied by $n^{O(\log k)}$. In Section 4, we show that in the worst case, the memory requirements are also multiplied by $n^{O(\log k)}$. However, if the system graph has bounded degree, then the memory cost is at worst $O(\log n)$. In Section 5, we obtain a polynomial time self-stabilizing algorithm for finding a maximal irredundant set, a problem which requires distance-4 information. In Section 6, we show how to efficiently obtain maximal \mathcal{P} -sets, for properties \mathcal{P} that are local.

A very natural problem in distributed computing is mutual exclusion to distance k , that is, getting exclusive access to all node states up to distance k . So if S is a local mutual exclusion algorithm such as in [1–3, 12], then one can amplify the distance of exclusion. Indeed, our algorithm itself can be viewed as a solution to resource allocation to distance k , but it does not deal with fairness and hence liveness. We think of it as more useful where the underlying algorithm ensures fairness, or where the underlying algorithm always terminates (it has a “static” goal). Recently, Danturi et al. [4] generalized the dining philosophers problem to avoid conflicts to distance k , and presented a deterministic solution.

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 $n > 1$.

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. We will use the problem of finding a maximal k -packing to illustrate our ideas; however we mention that recently, a better solution to maximal k -packing was provided by Manne and Mjelde [11].

Algorithm 1 is a well-known and simple self-stabilizing algorithm for finding the characteristic function of a maximal independent set [13]. The variable f can have two values, zero or one. It is easy to show that this algorithm stabilizes

in at most $2n$ moves in the usual distance-1 model, and furthermore that at stabilization the set $\{i : f(i) = 1\}$ is maximal independent.

Algorithm 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 The distance- k model

In [7], 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 [7] recursively.

We now define a class of self-stabilizing algorithm models. For each $k \geq 1$, let the *distance- k ball* at i , denoted $B^k[i]$, be the subgraph whose node set is $N^k[i]$, and which contains all edges e incident to nodes j , whose distance from i is *less* than k . Note this subgraph may not be the subgraph induced by $N^k[i]$. When $k = 1$, the graph $B^1[i]$ is a star. In the *distance- k self-stabilizing model*, each node i can instantly see its distance- k ball, along with all state information of these nodes. Included in this state information is the node's ID. Thus we may think of the information available to i , as a labeled graph, where the labels are node states. We refer to this as node i 's *distance- k information*. Note 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. Figure 1 shows three different views from node i under various models. Note, for example, that an edge between nodes b and y would be visible to node i in the distance-4 model, but not in the distance-2 model.

Now consider Algorithm 2, which assumes the distance-4 model, and uses a binary variable f . At each step the set $S = \{i \mid f(i) = 1\}$ defines a set of nodes.

Algorithm 2: 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 $f(i) = 0$

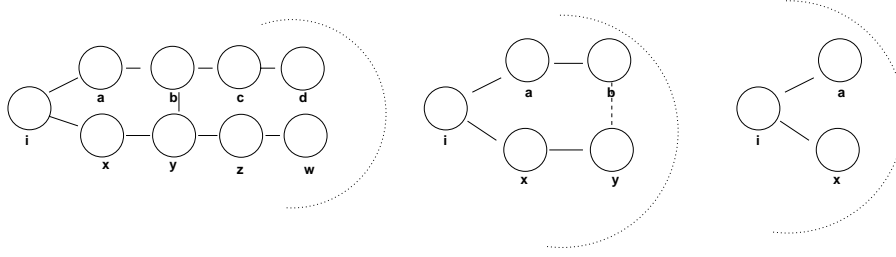


Fig. 1. Distance-4, distance-2, and distance-1 information.

Lemma 1. *Under the distance-4 model, Algorithm 2 finds a maximal 4-packing in at most $2n$ moves.*

Proof. We first claim that if Algorithm 2 stabilizes, the set $S = \{i \mid f(i) = 1\}$ must be a 4-packing. For if $i \in S$, there cannot be another $j \in N^4(i)$, or i would be privileged to execute a LEAVE. Therefore S is a 4-packing. Next, we claim that if the algorithm stabilizes, the 4-packing S is maximal. For if $S \cup \{i\}$ is also a 4-packing, i would be privileged to execute an ENTER. To complete the proof, we show that each node can make at most two moves. 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.

3 Conversion to distance-1

Assume now that we have some distance- $2k$ algorithm \mathcal{S}_{2k} , such as Algorithm 2, 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 the node's distance- k information. We may assume that $\sigma(i)$ is a graph where each node is labeled with a pair (j, f_j) , where j is an ID of a node in $N^k(i)$. We say that $\sigma(i)$ is *correct* if it represents node i 's distance- k information. In particular, this implies that for each $(j, f_j) \in \sigma$, $f(j) = f_j$.
- A pointer stores the ID of a member of $N^k[i]$, or has the value NULL. We write $i \rightarrow j$, $i \rightarrow i$, and $i \rightarrow \text{NULL}$ 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 $f(j')$.

For example, consider the leftmost graph of Figure 1, but assume only the distance-2 model. Then each node stores a σ containing f -values in its 2-neighborhood. Since node i can directly see node b , and since $\sigma(b)$ contains (c, f_c) and (d, f_d) , node i has an indirect view of c and d . However, it is possible for this view to be stale or 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

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

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

Algorithm \mathcal{S}_k is displayed as Algorithm 3. When $k = 1$, it is exactly the algorithm described in [7].

Algorithm 3: DISTANCE- k ALGORITHM \mathcal{S}_k

comment: simulates distance- $2k$ algorithm \mathcal{S}_{2k}

local variables: f, σ, \rightarrow

UPDATE- σ : if $\sigma(i)$ is incorrect
 then update $\sigma(i)$

ASK: if i is \mathcal{S}_{2k} -alive $\wedge (\forall j \in N^k[i] : j \rightarrow \text{NULL}) \wedge \sigma(i)$ is correct
 then $i \rightarrow i$

RESET: if $i \not\rightarrow \min N^k[i] \wedge \sigma(i)$ is correct
 then $i \rightarrow \min N^k[i]$

CHANGE: if $\forall j \in N^k[i] : j \rightarrow i \wedge \sigma(i)$ is correct
 then $\begin{cases} \text{if } i \text{ is } \mathcal{S}_{2k}\text{-alive, then update } f(i) \\ i \rightarrow \text{NULL} \end{cases}$

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 \mathcal{S}_k 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, $\min N^k[i]$ 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 node 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 pointing to NULL. For j to execute CHANGE, i must be pointing to j .

Lemma 4. *If node 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 pointing to 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. Assume that i makes the ASK move at time t_a , and makes its next CHANGE move at time t_c . Let j be some member of $N^k[i]$. At time t_a , j was pointing to NULL, and at t_c , j was pointing to i . So let t' , be the last time in the interval $[t_a, t_c]$ when j pointed to i . Clearly at t' , $\sigma(j)$ was correct. But throughout the interval $[t', t_c - 1]$, $\sigma(j)$ 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 node i makes a CHANGE move, then its next ASK move is correct.*

Proof. Assume that i makes a CHANGE move at time t_c , and its next ASK move is at time t_a . Let j be some member of $N^k[i]$. At time t_c , j was pointing to i , and at t_a , j was pointing to NULL. Let t' be the last time in the interval $[t_c, t_a]$ where j changed its pointer. Then clearly at t' , $\sigma(j)$ was correct. But $\sigma(j)$ must have remained correct throughout the interval $[t', t_a]$ since no member of $N^k[j]$ could have performed a CHANGE move if j 's pointer remained NULL.

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

Proof. If node i makes a RESET move at time t , and later makes another RESET move at time t' , $\min N^k[i]$ must have changed in the time interval between t and t' . Thus the set $\{j \mid j \in N^k[i] \wedge j \rightarrow j\}$ changed. This can happen only if some $j \in N^k[i]$ executes an ASK or CHANGE.

For convenience, we define a REAL-CHANGE move as a CHANGE move in which the variable f is updated. 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 node 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. By Lemma 8, during an interval with no *REAL-CHANGE* moves, each node i can make only $O(n)$ moves, since $d_i^k \leq n$. Thus the total number of moves made by the system is $O(n^2)$.

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 3 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. It is precisely the *REAL-CHANGE* moves that modify the variable f . Since there are no incorrect *REAL-CHANGE* moves, each *REAL-CHANGE* move is made by a node i because it is privileged with respect to \mathcal{S}_{2k} . Thus this subsequence of *REAL-CHANGE* moves could have been selected by the \mathcal{S}_{2k} daemon, and represents a valid computation.

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

Proof. By Lemma 11, there can be at most $a(n)$ *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 $a(n)$ moves can be implemented with a distance- k algorithm \mathcal{S}_k that stabilizes in $O(a(n)n^3)$ 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 $O(a(n)n^2)$ moves. Finally by Lemma 2, the algorithm is correct.

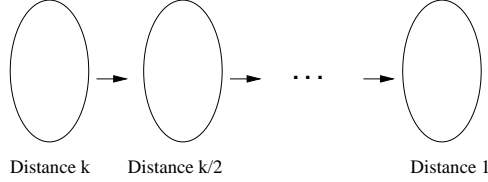


Fig. 2. Translating distance- k to distance-1.

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

Proof. Let us first assume k is a power of two. Then we may translate the distance- k algorithm S into a distance-1 algorithm by repeatedly halving the distance, as illustrated in Figure 2. From Theorem 1 it follows that the running time will be multiplied by a factor in $O(n^{3 \log_2 k})$, and so the result follows. Now assume $2^{i-1} < k < 2^i = j$. Note that we may run any distance- k algorithm under the distance- j model, since any information that nodes have at distance- k is preserved at distance- j . Since $\log_2 j < 2 \log_2 k$, we now can translate our algorithm to a distance-1 algorithm which runs in $a(n)n^{O(\log j)}$ moves, which is $a(n)n^{O(\log k)}$.

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

Proof. By Lemma 1 there is an $O(n)$ distance-4 algorithm. Using two translations, each costing $O(n^3)$, we obtain a distance-1 algorithm running in $O(n^7)$.

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 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 . For example, consider node i , shown in Figure 1. Under the distance-4 model, i can see the contents of nodes a, b, c, d and x, y, z, w . In the distance-2 model, i has only a direct view of a, b, x, y . Node i will store f , a pointer \rightarrow , and σ_i , where σ_i contains

$$(a, f_a), (b, f_b), (x, f_x), (y, f_y).$$

Since, node i can read the contents of nodes b and y , which contain σ_b and σ_y respectively, node i can read f_c, f_d, f_z, f_w . When this distance-2 algorithm

is translated to distance-1, node i will contain $(f, \rightarrow, \sigma_i)$, a pointer \rightarrow' , and σ'_i , where σ'_i contains

$$(a, (f_a, \rightarrow'_a, \sigma_a)), (b, (f_b, \rightarrow'_b, \sigma_b))$$

In the distance-1 model, node i can directly read the state of nodes a and x , which contain respectively σ'_a and σ'_x . Since σ'_x contains σ_b and σ'_x contains σ_y , node i achieves an indirect view of $B^4[i]$.

4 Memory overhead

We now consider the memory overhead involved in our translation. In our analysis we will assume that pointers and IDs require $\log n$ bits. Using the same transformation in the proof of Theorem 2, we may assume k is a power of two.

Theorem 3. *In a network with n nodes, where a largest $\frac{k}{2}$ -neighborhood has size t , a distance- k algorithm which uses b bits per node can be implemented by a distance-1 algorithm using storage $O(t^{\log k}(\log n + b + t))$.*

Proof. Let \mathcal{S}_k be a distance- k algorithm, in which $k = 2^r$. Translating it to a distance-1 algorithm \mathcal{S}_1 requires r steps, shown in Figure 2. Let m_i denote the memory requirements per node of the distance- $\frac{k}{2^i}$ created at stage i . Then $m_0 = b$. First consider the translation from \mathcal{S}_k to $\mathcal{S}_{\frac{k}{2}}$. Each node i in $\mathcal{S}_{\frac{k}{2}}$ will store a pointer, the variable f , and σ , which represents the labeled graph $B^{\frac{k}{2}}[i]$. This memory overhead is dominated by σ , representing a labeled graph having at most t nodes and at most t^2 edges. Each node in the graph will be labeled with an address v and value f_v . Hence it will require storage proportional to $m_1 = t(\log n + b) + t^2$. Similarly, for $i \geq 1$,

$$m_{i+1} \leq t(\log n + m_i) + t^2 \quad (1)$$

We claim that for each $i \geq 1$,

$$m_i \leq it^i(\log n + b + t) \quad (2)$$

As observed already, (2) holds when $i = 1$, so by induction, assume the relation holds for some positive i . Then using (1) we get

$$\begin{aligned} m_{i+1} &\leq t(\log n + m_i) + t^2 \\ &\leq t(\log n + it^i(\log n + b + t)) + t^2 \\ &= t \log n + it^{i+1}(\log n + b) + it^{i+1} + t^2 \\ &\leq (i+1)t^i(\log n + b) + (i+1)t^{i+1} \\ &= (i+1)t^i(\log n + b + t) \end{aligned}$$

This completes the induction. Setting $i = \log k$ in (2) we are done, since k is constant.

Corollary 2. *Any distance- k algorithm that uses b bits per node, where b is constant or polynomial in n , can be translated to a distance-1 algorithm that uses storage $n^{O(\log k)}$.*

Proof. Replacing t in the expression above with n , we get $(\log n + b + n)n^{\log k}$. But $\log n + b + n$ is polynomial in n , so this becomes $n^{O(\log k)}$.

Corollary 3. *If the network has bounded degree, then a distance- k algorithm that uses b bits per node can be translated to a distance-1 algorithm which uses $O(\log n + b)$ storage per node.*

Proof. The numbers t and $\log k$ are constants.

5 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* [10] 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. Although minimal dominating sets are maximal irredundant, there exist maximal irredundant sets that are not minimal dominating. We would like a general algorithm for maximal irredundant sets, that is, an algorithm that can potentially identify any maximal irredundant set. Finding such an algorithm 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 4. 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 4: 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 the set $\{f(j') \mid j' \in N^2[j]\}$.

Theorem 4. *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 4 can be implemented in the distance-4 model. By our earlier comments, Algorithm 4 stabilizes in a linear number of moves. The analysis follows by Theorem 2.

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

6 Local monotonic properties

The ideas in Section 5 can be generalized. Let k be a positive integer. For $i \in V$ and $S \subseteq V$, let G_i^k be the subgraph induced by $N^k[i]$, and let $S_i^k = N^k[i] \cap S$. Many properties \mathcal{P} that describe vertex sets S are *local*. By this we mean there is a predicate p that depends on $i \in V$, G_i^k , and S_i^k . S has \mathcal{P} if and only if for all $i \in S$, $p(i, G_i^k, S_i^k)$ is true. For example, a set S is independent if and only if for all $i \in S$, $|N[i] \cap S| = 1$. We will write $p(i)$ to mean $p(i, G_i^k, S_i^k)$.

We will also insist that p is *monotonic*. That is, for any j , $p(i, G_i^k, S_i^k - \{j\})$ is true whenever $p(i, G_i^k, S_i^k)$ is true. Informally, this means that if $p(i)$ is true, then it will remain true if some j is removed from S . We will refer to such properties as *local monotonic*.

Recall that a property \mathcal{P} is *hereditary* if whenever S has \mathcal{P} , all subsets of S also have \mathcal{P} . It is easy to see that local monotonic properties are hereditary. Recall also that a \mathcal{P} -set S is said to be *1-maximal* if, for all x , no proper superset $S \cup \{x\}$ has \mathcal{P} . In general, if a set is 1-maximal, it may not be maximal. However, it is easy to see for hereditary properties \mathcal{P} , a set S is maximal if and only if it is 1-maximal.

We wish to generalize some of the algorithms considered earlier, including Algorithm 1, Algorithm 2, and Algorithm 4. Let us say that a node $i \notin S$ is *safe* provided i entering S would make $p(i)$ true and preserve all true $p(j)$, $j \in N^k(i)$. Formally, i is safe if

1. $p(i, G_i^k, S_i^k \cup \{i\})$ is true; and

2. for all $j \in N^k(i)$, $p(j, G_j^k, S_j^k) \Rightarrow p(j, G_j^k, S_j^k \cup \{i\})$.

A node i can determine if it is safe by distance- $2k$ information. Now consider Algorithm 5, which is a distance- $2k$ algorithm.

Algorithm 5: MAXIMAL \mathcal{P} -SET, \mathcal{P} A LOCAL MONOTONIC PROPERTY

local variable: f
ENTER: if $f(i) = 0 \wedge i$ is safe
 then $f(i) = 1$
LEAVE: if $f(i) = 1 \wedge \neg p(i)$
 then $f(i) = 0$

Theorem 5. *Let \mathcal{P} be a local monotonic property. In the distance- $2k$ model, Algorithm 5 finds a maximal \mathcal{P} -set in $2n$ moves.*

Proof. If Algorithm 5 stabilizes, then since no node can LEAVE, $p(i)$ is true for all $i \in S$, and so S is a \mathcal{P} -set. Finally we see that since no node can ENTER, S is 1-maximal and therefore maximal. Next we claim the system must stabilize in $2n$ moves. For when a node enters S , $p(i)$ becomes true. This will remain true, because monotonicity prevents leaving nodes from changing $p(i)$, and safety prevents entering nodes from changing $p(i)$. Therefore once a node executes ENTER, it will never move again. It follows that no node can move more than twice.

Corollary 4. *For any local monotonic property \mathcal{P} , there is a self-stabilizing algorithm for finding a maximal \mathcal{P} -set in a polynomial number of moves.*

Proof. By Theorem 5, there is a linear-time distance- $2k$ algorithm to find a maximal \mathcal{P} -set. By Theorem 2 there is a distance-1 algorithm which runs in $nn^{O(\log k)}$ moves.

As a final note about local monotonic properties \mathcal{P} , one can easily show that with any such property, a maximal \mathcal{P} -set can be obtained using a one-pass greedy algorithm.

7 Conclusions and future work

The contribution of this paper is a general methodology for transforming any distance- k algorithm, such as Algorithm 4, into a self-stabilizing algorithm. Applications include finding maximal k -packings, maximal irredundant sets and maximal \mathcal{P} -sets for any local monotonic property \mathcal{P} . The time and space overhead of this transformation is in $n^{O(\log k)}$. We suggest as further research trying to improve this time and memory overhead.

References

1. J. Beauquier, S. Cordier and S. Delaët, Optimum probabilistic self-stabilization on uniform rings, Proceedings of the Second Workshop on Self-Stabilizing Systems, 1995 15.1–15.15.
2. C. Boulinier, F. Petit and V. Villain, When graph theory helps self-stabilization, PODC '04: Proceedings of the Twenty-third Annual ACM Symposium on Principles of Distributed Computing, St. John's, 2004, 150–159, ACM Press, New York.
3. S. Cantarell, A.K. Datta, F. Petit and V. Villain, Token based group mutual exclusion for asynchronous rings 21st International Conference on Distributed Computing Systems (ICDCS), 2001. Apr 2001 Mesa, 691-694.
4. Praveen Danturi, Mikhail Nesterenko and Sebastien Tixeuil, Self-Stabilizing Philosophers with Generic Conflicts, 8th International Symposium Stabilization, Safety, and Security of Distributed Systems, SSS 2006, Dallas, November, 2006 Lecture Notes in Computer Science , Vol. 4280 Datta, Ajoy K.; Gradinariu, Maria (Eds.).
5. E. W. Dijkstra, Self-stabilizing systems in spite of distributed control, Comm. ACM **17** (11) (1974) 643–644.
6. S. Dolev, *Self-Stabilization*. MIT Press, 2000.
7. M. Gairing, W. Goddard, S.T. Hedetniemi, P. Kristiansen and A.A. McRae, Distance-two information in self-stabilizing algorithms, Parallel Process. Lett., **14** (2004) 387–398.
8. W. Goddard, S.T. Hedetniemi, D.P. Jacobs and P.K. Srimani, Self-stabilizing global optimization algorithms for large network graphs, Int. J. Dist. Sensor Net., **1** (2005) 329–344.
9. W. Goddard, S.T. Hedetniemi, D.P. Jacobs and V. Trevisan, Distance- k information in self-stabilizing algorithms, in 13th Colloquium on Structural Information and Communication Complexity (SIROCCO), Chester, July 2006, LNCS 4056, 349–356.
10. T.W. Haynes, S.T. Hedetniemi and P.J. Slater, *Fundamentals of Domination in Graphs*, Marcel Dekker, New York, 1998.
11. F. Manne and Morten Mjelde, A memory efficient self-stabilizing algorithm for maximal k -packing, 8th International Symposium Stabilization, Safety, and Security of Distributed Systems, SSS 2006, Dallas, November, 2006 Lecture Notes in Computer Science , Vol. 4280 Datta, Ajoy K.; Gradinariu, Maria (Eds.).
12. M. Nesterenko and A. Arora, Stabilization-preserving atomicity refinement, Journal of Parallel and Distributed Computing, **62** , 5 (2002), 766 - 791.
13. S.K. Shukla, D.J. Rosenkrantz and S.S. Ravi, Observations on self-stabilizing graph algorithms for anonymous networks, Proceedings of the Second Workshop on Self-Stabilizing Systems, 7.1–7.15, 1995.