A Silent Self-Stabilizing Algorithm for the Generalized Minimal k-Dominating Set Problem*

Ajoy K. Datta Stéphane Devismes Lawrence L. Larmore

Abstract

We give a silent self-stabilizing algorithm for the *generalized minimal k-dominating set* problem in a connected distributed network G. Given a positive integer k and two sets of processes $R^* \subseteq D^*$, the problem is to find a set D which is minimal subject to the conditions that $R^* \subseteq D \subseteq D^*$ and that D is k-dominating relative to D^*, meaning that every process within distance k of D^* is also within distance k of D. The algorithm is order-invariant, requires $O(\log N + \log k)$ space per process, works under the distributed unfair scheduler, and converges in $O(N + k)$ rounds, where N is a given upper bound on the number of processes.

Keywords: dominating set, self-stabilization, unison, proof labeling scheme.

1 Introduction

Self-stabilization [2, 3] is a property of a distributed algorithm, which enables that algorithm to withstand transient faults in the system. A distributed algorithm is *self-stabilizing* if, after transient faults hit the system and place it in some arbitrary global state, the system recovers without any external (*e.g.,* human) intervention in finite time.

In this paper, we consider the problem of computing a *k-dominating set* of a network in a self-stabilizing manner. For any $k \geq 1$, a set D of nodes of a graph G is defined to be *k-dominating* if every node of G is within distance k of some member of D.[1] The problem of computing a k-dominating set is related to *k-clustering*, which consists of partitioning the network into subgraphs, called *clusters*, such that in each cluster, one process, called the *clusterhead*, is designated to be the leader of the cluster and each member of the cluster is within distance k of the clusterhead. The set of clusterheads of a given k-clustering is a k-dominating set; conversely, if D is a k-dominating set, a k-clustering can be constructed for which D is the set of clusterheads.

Ideally, we would like to compute a *minimum* k-dominating set, namely a k-dominating set of the smallest possible cardinality. However, that problem is known to be $\mathcal{NP}$-hard [13]. Instead, we focus on the tractable problem of finding a *minimal k-dominating set*, *i.e.,* a k-dominating set which has no proper subset which is also k-dominating. We can envision applications where only vertices which are members of a given set can be chosen to be members of D, and also where all members of some smaller set are *required* to be members of D. In this paper, we consider the following generalization of the k-dominating set problem: we are given an integer k and a graph G as before, and we are also given sets of nodes $R^* \subseteq D^*$. A solution to the generalized problem is then a set D such that $R^* \subseteq D \subseteq D^*$ and D is k-dominating *relative to* D^*, meaning that every node which is within distance k of D^* is also within distance k of D and D is minimal subject to those conditions. In an application, the members of D might have more demanding tasks than other processes. D^* might be the set of processes strong enough to serve as dominators, while R^* is a set of processes which

*A preliminary version of this work was presented in SRDS 2009 [1].

[1] Our definition of k-dominance is the one used in most of the related work [1, 4, 5, 6, 7, 8, 9, 10, 11]. However, it differs from the definitions given by some other authors, such as Kuhn *et al.* [12]. The notion of k-dominating set used in [12] defines a different problem, which is computing a subset of nodes S such that every node (of G) not in S has at least k neighbors in S.

are especially strong, and hence must be chosen to be dominators. The original k-dominating set problem is then the special case that $D^* = G$ and $R^* = \emptyset$. We note that, even in the case that D^* is k-dominating, a solution to the generalized minimal k-dominating set problem may not itself be a minimal k-dominating set. Figure 1.1 illustrates such a case.

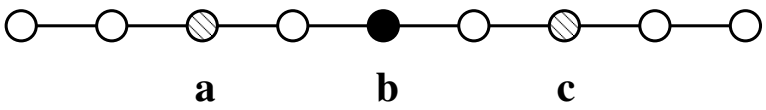


Figure 1.1: Instance of the generalized minimal k-dominating set problem where D^* is k-dominating, yet there is no minimal k-dominating solution. G is a chain of nine nodes, $k = 2$, $D^* = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$, and $R^* = \{\mathbf{b}\}$. The only solution to the generalized problem is $D = D^*$, which is not minimal k-dominating (indeed, $\{\mathbf{a}, \mathbf{c}\}$ is k-dominating).

Detailed Contributions In this paper, we give a distributed self-stabilizing algorithm, GMD, for the generalized minimal k-dominating set problem, for given G, k, D^*, and R^*.

We assume that processes have unique identifiers. GMD is *order-invariant* in the sense of Naor and Stockmeyer [14, 15]; that is, our algorithm would behave the same way if each identifier, $x.id$, were replaced by a new identifier, $x.id'$, such that $x.id' < y.id'$ if and only if $x.id < y.id$.

GMD assumes a knowledge of a given upper bound N on the number of processes and is implemented in the *locally shared memory model with composite atomicity* introduced in [2]. GMD has round complexity $O(N + k)$ and space complexity $O(\log N + \log k)$ per process. We use a *proof labeling scheme* [14] to detect legitimacy, allowing the algorithm to end the computation when legitimacy is satisfied: GMD is silent. The construction of the generalized minimal k-dominating set is essentially independent of the construction of the labeling.

Related Work To the best of our knowledge there was no self-stabilizing distributed algorithm to construct a minimal k-dominating set of a network prior to our submission of this paper, although a recent paper by Johnen [4] addresses the problem. In particular, note that the silent self-stabilizing algorithm given in [16] uses the preliminary version of our algorithm (published in the conference SRDS 2009 [1]) as a module. Specifically stated, the algorithm in [16] first computes a not-necessarily minimal k-dominating set of at most $\lceil \frac{n}{k+1} \rceil$ processes, then uses our algorithm to minimize this set. This illustrates the versatility of our technique. Moreover, the experiments given in [16] show that in most cases, we obtain an appreciable gain when using our algorithm to minimize a k-dominating set. Note that the algorithm proposed in [16] stabilizes in $O(n)$ rounds using $O(\log k + \log n + k \log \frac{N}{k})$ bits per process, where n is the number of process (resp. N is an upper bound on n). These latter complexities show that using our algorithm as a final module to minimize a k-dominating set induces an overhead that is usually asymptotically negligible.

There are several (silent) self-stabilizing distributed algorithms that construct k-dominating sets which may not be minimal, *e.g.,* [5, 6, 7, 8]. The solutions given in these papers are silent, order-invariant, and are implemented in the locally shared memory model with composite atomicity, assuming an arbitrary network with unique identifiers. The algorithm given in [5] stabilizes in $O(kn)$ rounds using $O(\log k + \log n)$ space per process. The algorithm given in [7] stabilizes in $O(k)$ rounds using $O(k \log n)$ space per process. The algorithm of [6] stabilizes in $O(kn)$ rounds using $O(k \log n)$ space per process. The algorithm given in [8] computes a k-dominating set of at most $\lceil \frac{n}{k+1} \rceil$ processes. It stabilizes in $O(n)$ rounds using $O(\log k + \log n)$ space per process. Finally, for Unit Disk Graphs, the k-dominating set computed by the algorithm is $7.2552k + O(1)$-*competitive*, *i.e.,* the size of the k-clustering computed by the algorithm is at most $7.2552k + O(1)$ times as large as the minimum solution.

There are several known (silent) self-stabilizing solutions for the minimal 1-dominating set problem, *e.g.,* [17, 18, 19]. In [19], the authors gave two self-stabilizing algorithms which find a maximal independent

set, which is necessarily a minimal 1-dominating set. The first solution is deterministic and assumes a central scheduler, *i.e.,* just one process executes an action in each step. The second solution is randomized and assumes a distributed scheduler. The self-stabilizing algorithm of Ikeda *et al.* [17] is deterministic, assumes a synchronous network with unique identifiers, and is optimal in space. Another deterministic self-stabilizing solution for the maximal independent set problem is given in [18]. The claimed time complexity of their solution is $O(diam)$ rounds, where $diam$ is the diameter of the network. However, careful examination of the algorithm reveals that its actual time complexity is $\Omega(n)$ rounds even if the diameter is small, as we showed in [20].

In [21], Drabkin *et al* gave several (silent) self-stabilizing algorithms to compute a 1-dominating *connected* set in an asynchronous system. The computed solutions are not necessarily minimal. All proposed algorithms stabilize in $O(n)$ rounds using $O(\log n)$ space per process. In [22], Kamei and Kakugawa gave a deterministic (silent) self-stabilizing algorithm which computes an approximation of a minimum 1-dominating connected set in a synchronous system for a unit disk graph and stabilizes in $O(n)$ steps.

There are several known (silent) self-stabilizing solutions for the 1-clustering problem, *e.g.,* [23, 24]. The algorithm of Mitton *et al.* [24] is restricted to wireless sensor networks. Johnen and Nguyen gave a self-stabilizing 1-clustering algorithm such that each clusterhead is a neighbor of at most m other clusterheads for some given $m \geq 1$ [23].

Notice that all solutions proposed in [24, 21, 22, 23] are implemented in the locally shared memory model with composite atomicity.

There are several deterministic, non-self-stabilizing, distributed solutions for finding a (not necessary minimal) k-clustering of a network [9, 10, 11], all written in the message-passing model. The solution given in [10] assumes a synchronous system and has time complexity $O(k \log^* n)$. The solutions given in [9, 11] assume an asynchronous system, and their time complexities are not given. The algorithms given in [9, 10] compute a k-clustering of size $O(n/k)$ in an arbitrary connected network. Moreover, if the network is a unit disk graph, the algorithm of [9, 11] is $O(k)$-competitive. Finally, the algorithm given by Spohn and Garcia-Luna-Aceves [11] solves another generalized version of the k-clustering problem. In this version, a parameter m is given, and each process must be a member of m different k-clusters.

Outline of the Paper In Section 2, we define the *locally shared memory with composite atomicity* computational model, self-stabilization, and silence. In Section 3, we give a formal definition of GMDS, the *generalized minimal k-dominating set* problem, and prove an upper bound on the size of any solution to that problem. In Section 4, we give a distributed algorithm for the *up to M-nearest member* problem. This algorithm finds the distances to the closest and second closest members of a given set Z to each process provided those distances do not exceed M, and is used a number of times by GMD. In Section 5, we explain the theory of *proof labels*, as defined by Korman *et al.* [14], and also construct a proof labeling scheme for the GMDS problem. In Section 6, we give an abstract algorithm, ABST, for the GMDS problem. ABST solves the generalized minimal k-dominating set problem in $O(n/k)$ steps from any starting state. ABST is not distributed, *i.e.,* it assumes global scope. We follow that construction by defining a slightly less abstract algorithm, SYNC, which emulates each step of ABST in $2k+3$ steps. In Section 7, we define GMD. We first give an overview, then define two modules: the unison $\mathcal{U}$ (the necessary assumption on the knowledge of N is due to this latter module) and the array update module $\mathcal{D}$. Finally, we give the formal definition of GMD, which is constructed using those two modules together with the *proof labeling scheme* $\mathcal{L}$ defined in Section 5 and the *nearest member* algorithm $\mathcal{Z}$ given in Section 4. In Section 8, we prove the self-stabilization and silence of GMD, and then compute its round complexity. Our main result is given as Theorem 8.1. We give concluding remarks in Section 9. To help the reader, we also give a table of symbols in pages 38-38.

2 Preliminaries

We consider a bidirectional network of n processes, where every process x has a unique identifier, $x.id$. If b bits are used to store each identifier, then the space complexity of our algorithm will be $\Omega(b)$ per process. Henceforth, as is common in the literature, we will assume that $b = \Theta(\log n)$. We represent a network

as an undirected connected graph G, where the nodes of G are the processes, and the edges of G are the communication links between processes.

2.1 Computational Model

We use the *locally shared memory model* of computation [2], where a process x can read its own variables and those of its neighbors, but can write only to its own variables. Each process operates according to its (local) *program*. A *distributed algorithm* $\mathcal{A}$ is a collection of n programs, each executed by one process. The program of a process consists of a finite set of *actions*. Each action has two components: a *guard* and a *statement*. The *guard* of an action in the program of a process x is a Boolean expression involving the values of the variables of x and its neighbors. The *statement* of an action of x updates one or more variables of x. An action can be executed only if it is *enabled*, *i.e.,* its guard evaluates to true. A process is said to be enabled if at least one of its actions is enabled.

The *state* of a process in $\mathcal{A}$ is defined by the values of its variables. A *configuration* of $\mathcal{A}$ is a vector consisting of one state of each process. A configuration γ of $\mathcal{A}$ is said to be *final* if no process is enabled at γ.

A *step* $\gamma \mapsto \gamma'$ consists of one or more enabled processes *atomically* executing an action; this model is called *composite atomicity* [3]. Each step from a configuration to another is driven by a *scheduler*, also called a *daemon*. We assume that the daemon is *distributed*: if one or more processes are enabled, the daemon selects at least one of the enabled processes to execute an action. We also assume that daemon is *unfair*, *i.e.,* that it need never select a given process unless it becomes the only enabled process.

We define a *computation* to be a *maximal* sequence of configurations $\gamma_0\gamma_1\ldots$ such that each $\gamma_i \mapsto \gamma_{i+1}$ is a step. By maximal, we mean that the computation is infinite, or that it is finite and its last configuration is final. We use a superscript of t to denote the value of any quantity at γ_t.

We say that a process x is *neutralized* in the step $\gamma_i \mapsto \gamma_{i+1}$ if x is enabled in γ_i and not enabled in γ_{i+1}, but does not execute any action at that step. The neutralization of a process is caused by one or more neighbors of x changing their states, resulting in the guards of all actions of x becoming false.

To measure time complexity, we use the notion of *round*, introduced by Dolev *et al* [25]. The first round of a computation Γ is the minimal prefix of Γ in which every process that is enabled in the initial configuration either executes an action or is neutralized. Let Γ' be the suffix of Γ starting from the last configuration of that first round. The second round of Γ is the first round of Γ', third round of Γ is the second round of Γ', and so forth.

2.2 Self-Stabilization and Silence

Let $\mathbb{P}$ be any predicate on configurations of $\mathcal{A}$. We say $\mathcal{A}$ *self-stabilizes* to $\mathbb{P}$ if there exists a set of configurations of $\mathcal{A}$, which we call the *legitimate* configurations, such that:
Correctness. Every legitimate configuration satisfies $\mathbb{P}$.
Closure. Every step of $\mathcal{A}$ starting from a legitimate configuration yields a legitimate configuration.
Convergence. Every computation of $\mathcal{A}$ contains a legitimate configuration.
We say that $\mathcal{A}$ is *silent* if every computation of $\mathcal{A}$ is finite. By definition, $\mathcal{A}$ is silent and self-stabilizing *w.r.t.* $\mathbb{P}$ if the following two conditions hold: (1) all computations of $\mathcal{A}$ are finite; and (2) all final configurations of $\mathcal{A}$ satisfy $\mathbb{P}$.

Rounds and the Unfair Daemon An algorithm with bounded round complexity might still never converge under the unfair daemon, since the last round will never end if the daemon ignores an enabled process. For that reason, a proof that the round complexity of an algorithm is finite does not prove it converges.

2.3 Limiting Properties and Proofs

If $\mathbb{P}$ is a property of a given algorithm $\mathcal{A}$ on a given network G, *i.e.,* a predicate which is defined for every configuration γ of $\mathcal{A}$ on G, we say that $\mathbb{P}$ holds *eventually* in an infinite computation Γ of $\mathcal{A}$ if there is some

step of Γ after which all configurations of Γ have property $\mathbb{P}$. We say that $\mathbb{P}$ holds *frequently* in an infinite computation Γ of the algorithm if, given any step of Γ, there is a subsequent configuration of Γ at which $\mathbb{P}$ holds.

Similarly, if A is an action of $\mathcal{A}$ and x is a process of G, we say that x executes A *frequently* during an infinite computation Γ if x executes A at infinitely many steps of Γ.

By definition, we have the following basic properties gathered in Remark 2.1 below.

Remark 2.1 *Suppose Γ is an infinite computation of an algorithm $\mathcal{A}$ in a network G.*

(a) *If $\mathbb{P}$ is a predicate in configurations of $\mathcal{A}$ in G, and if Γ is an infinite computation in which $\mathbb{P}$ holds eventually, then there is some suffix of Γ in which $\mathbb{P}$ holds in every configuration.*

(b) *If $\mathbb{P}$ is a predicate in configurations of $\mathcal{A}$ in G, and if Γ is an infinite computation of $\mathcal{A}$ in which $\mathbb{P}$ does not hold frequently, then there is some suffix of Γ in which $\mathbb{P}$ never holds.*

(c) *If A is an action of $\mathcal{A}$ and x is a process of G, and if Γ is an infinite computation of $\mathcal{A}$ in which x does not execute A frequently, then there is some suffix of Γ in which x never executes A.*

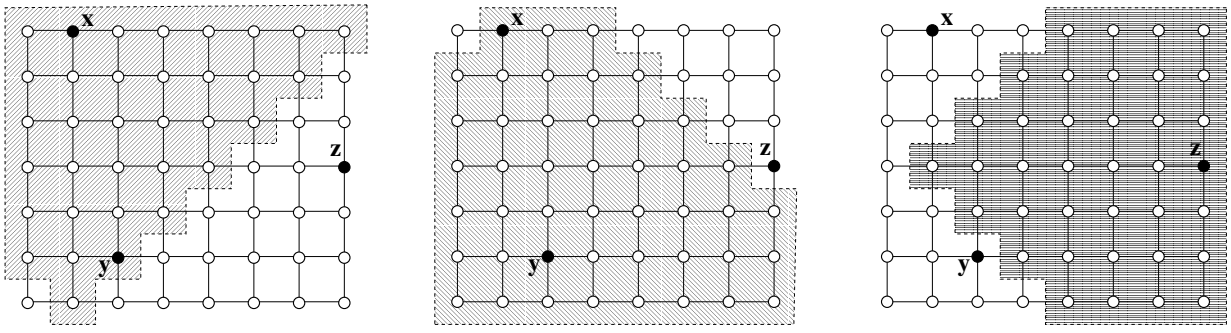
3 The Generalized Minimal k-Dominating Set (GMDS) Problem

In this section, we first discuss the minimal k-dominating set problem, and then introduce our generalization.

3.1 Notation

If x is a node of a graph, we let $N(x)$ denote the set of neighbors of x. If x, y are nodes, let $||x, y||$ be the distance from x to y, defined to be the minimum number of edges of any path from x to y. For any node x and any $d \geq 0$, let $N_d(x) = \{y : ||x, y|| \leq d\}$, the *$d$-hop neighborhood* of x. Note that $N_1(x) = N(x) \cup \{x\}$. If S is any set of nodes, we define $N_d(S) = \bigcup \{N_d(x) : x \in S\}$. If x is a node and S is a set of nodes, let $||x, S|| = \min \{||x, s|| : s \in S\}$. We let $||x, \emptyset|| = \infty$. For any node x and any set of nodes S let $||x, S||_2$ be the *secondary distance* of x to S, the distance to the *second* closest member of S. Formally, if $||x, S|| = ||x, y||$ with $y \in S$ then $||x, S||_2 = ||x, S \setminus \{y\}||$. If D is a set of nodes, $x \in D$, and $y \in N_k(x)$, we say that y is a *k-exclusive follower of x relative to D*, if $N_k(y) \cap D = \{x\}$. Let $Excl\_Followers_{k,D}(x)$ be the set of all k-exclusive followers of x relative to D. We will write $Excl\_Followers_D(x)$ in short if k is understood, and simply $Excl\_Followers(x)$ if both k and D are understood. We say a set of nodes D is *k-exclusive* if $Excl\_Followers(x)$ is non-empty for all $x \in D$.

Figure 3.1 illustrates k-dominance and k-exclusivity. In the figure, we consider the set of nodes $D = \{\mathbf{x}, \mathbf{y}, \mathbf{z}\}$ (shown in black) in a Grid network, and we let $k = 6$. The k-hop neighborhood of each node in D is indicated by a shaded background. Note that every node lies in at least one of the three k-hop neighborhoods; thus D is k-dominating. Note also that each of the k-hop neighborhoods contains at least one node not in either of the others; thus D is k-exclusive.



$\mathbf{H_k(x)}$ $\mathbf{H_k(y)}$ $\mathbf{H_k(z)}$

Figure 3.1: Example on a Grid graph.

3.2 The Minimal k-Dominating Set Problem

The input of an instance of the minimal k-dominating set problem is an ordered pair (G, k), such that G is a graph and k is an integer. The output is a k-dominating set of nodes D such that no proper subset of D is k-dominating. The following lemma connects the property of exclusivity with the minimal k-dominating set problem.

Lemma 3.1 *A k-dominating set D is minimal if and only if D is k-exclusive.*

Proof: Suppose D is not minimal. Then D has a proper subset $\widetilde{D}$ which is k-dominating. Pick $x \in D \setminus \widetilde{D}$. Then $N_k(D \setminus \{x\}) \supseteq N_k(\widetilde{D}) = N_k(D)$, which is the set of all nodes, and thus D is not k-exclusive.

Conversely, suppose D is not k-exclusive. Pick $x \in D$ such that $\mathit{Excl\_Followers}(x) = \emptyset$. $\forall y \in N_k(x)$, we have $N_k(y) \cap D \neq \{x\}$. Now, as D is k-dominating, we have $\forall y \in N_k(x)$, $N_k(y) \cap D \neq \emptyset$. Consequently, $N_k(y) \cap (D \setminus \{x\}) \neq \emptyset$, $\forall y \in N_k(x)$. Hence, $D \setminus \{x\}$ is also a k-dominating set, and thus D is not minimal. □

3.3 Specification of the Generalized Problem

An instance of the *generalized minimal k-dominating set* problem is an ordered quadruple (G, k, R^*, D^*) where G is a graph, k is an integer, and $R^* \subseteq D^* \subseteq G$. (In an application, we might have the additional condition that D^* is k-dominating.)

A *solution* to that instance is a set of nodes D, which is minimal subject to the *relative k-dominance condition*, $N_k(D) = N_k(D^*)$, and the *inclusion condition*, $R^* \subseteq D \subseteq D^*$. If D^* is k-dominating, the relative k-dominance condition is equivalent to the condition that D is k-dominating.

However note that, even in the case that D^* is k-dominating, a solution to the generalized minimal k-dominating set problem may not itself be a minimal k-dominating set. Figure 1.1 (page 2) illustrates such a case.

Lemma 3.2 *Given an input instance (G, k, R^*, D^*) of the generalized minimal k-dominating set problem, D is a solution to that instance if and only if*

(i) $R^* \subseteq D \subseteq D^*$,
(ii) $N_k(D) = N_k(D^*)$, *and*
(iii) $\mathit{Excl\_Followers}_{k,D}(x) \neq \emptyset$ *for any* $x \in D \setminus R^*$.

Proof: Similar to the proof of Lemma 3.1. Suppose D is a solution. Then (i) and (ii) hold trivially.

If $x \in D \setminus R^*$ and $\mathit{Excl\_Followers}_{k,D}(x) = \emptyset$, let $\widetilde{D} = D \setminus \{x\}$. Then $N_k(\widetilde{D}) = N_k(D) = N_k(D^*)$ and $R^* \subseteq \widetilde{D} \subsetneq D \subseteq D^*$, contradicting the minimality of D.

Conversely, suppose D satisfies (i), (ii), and (iii), and D is not minimal subject to those conditions. Then there is some $R^* \subseteq \widetilde{D} \subsetneq D$ where $N_k(\widetilde{D}) = N_k(D^*)$. Pick $x \in D \setminus \widetilde{D}$. By (iii), there is some node $y \in \mathit{Excl\_Followers}_{k,D}(x)$. Then $\widetilde{D}$ is not k-dominating relative to D^* since $y \notin N_k(\widetilde{D})$, contradiction. □

3.4 An Upper Bound on the Size of a k-Dominating Set

In Lemma 3.5, we prove an upper bound on the cardinality of any k-exclusive set, and hence any minimal k-dominating set, by Lemma 3.1. We first give some definitions and basic technical results.

Assume we are given k, a set D of nodes of a graph G, and $x \in D$. Let $\mathit{Closest}_D(x)$ be the set of all nodes in the graph which are strictly closer to x than to any other node of D. The subscript D can be omitted if it is understood. Note that $\mathit{Excl\_Followers}_{k,D}(x) \subseteq \mathit{Closest}_D(x)$. Figure 3.2 illustrates the sets $\mathit{Closest}(x)$ and $\mathit{Excl\_Followers}(x)$ for all $x \in D$ in a simple grid graph. In this example $k = 6$, and the members of $D = \{\mathbf{x}, \mathbf{y}, \mathbf{z}\}$ are shown in black.

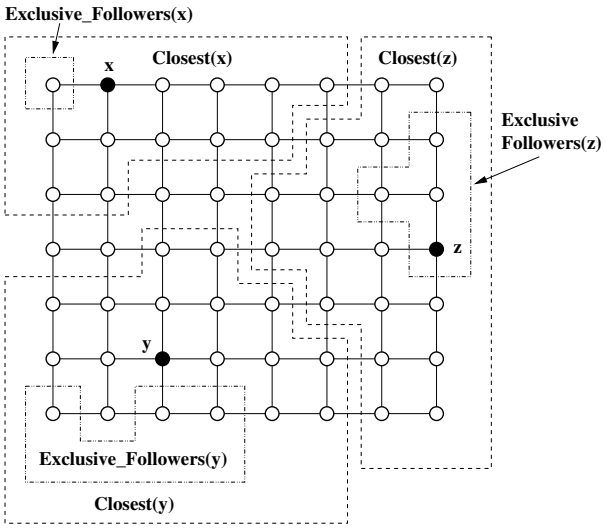


Figure 3.2: Example showing *Closest* and *Excl_Followers* in a Grid graph.

Lemma 3.3 *Suppose that p, q, r, and s are points in a metric space, such that s lies between p and q,* i.e., $||p,q|| = ||p,s|| + ||s,q||$. *Then*
(a) *if* $||p,q|| \leq ||p,r||$ *then* $||s,q|| \leq ||s,r||$,
(b) *if* $||p,q|| < ||p,r||$ *then* $||s,q|| < ||s,r||$.

Proof: We only prove (a); the proof of (b) is similar. By the triangle inequality, we have $||p,r|| \leq ||p,s|| + ||s,r||$. So, if $||p,q|| \leq ||p,r||$ then $||p,q|| \leq ||p,s|| + ||s,r||$ and, so $||p,s|| + ||s,q|| \leq ||p,s|| + ||s,r||$, *i.e.*, $||s,q|| \leq ||s,r||$. □

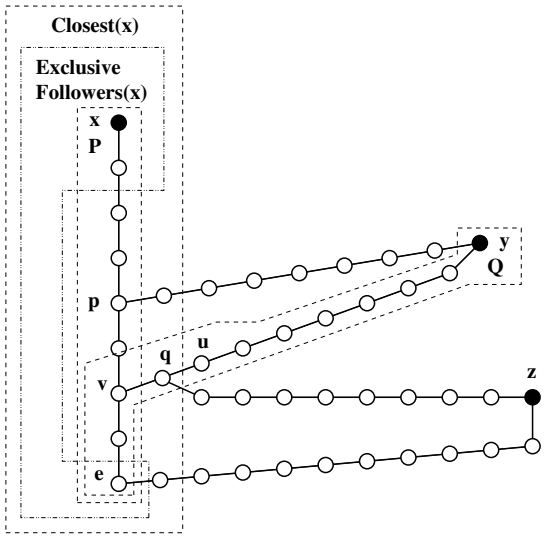


Figure 3.3: An example illustrating Claims I and II in the proof of Lemma 3.4. In this example, $k = 10$, $D = \{\mathbf{x}, \mathbf{y}, \mathbf{z}\}$, $a = ||\mathbf{x}, \mathbf{v}|| = 6$, $b = ||\mathbf{e}, \mathbf{v}|| = 2$, $c = ||\mathbf{y}, \mathbf{v}|| = 9$, and $d = ||\mathbf{u}, \mathbf{v}|| = 2$. The members of D are shown as black dots.

Lemma 3.4 *If D is a k-exclusive set of at least two nodes in a connected graph, and $x \in D$, then $Closest_D(x)$ has cardinality at least $\lceil \frac{k+1}{2} \rceil$.*

Proof: Pick $e \in Excl\_Followers_D(x)$ and a shortest path from x to e. Let P be the set of nodes on that path.
Claim I: $P \subseteq Closest_D(x)$.

Proof of Claim I: Let $p \in P$, and suppose that $p \notin Closest(x)$. Then, there exists $y \in D$, $y \neq x$, such that $||p, y|| \leq ||p, x||$. Since e is an exclusive follower of x, $||e, y|| > ||e, x||$. By Lemma 3.3(b), $||p, x|| < ||p, y||$, contradiction. Thus, $p \in Closest(x)$.

Let y be the second closest element of D to e. Pick a shortest path of length greater than k from y to e, and let Q be the set of nodes on that path. Let u be the node in Q farthest from y such that $||y, u|| \leq ||x, u||$, and let v be the node in $P \cap Q$ which is farthest from e. (Figure 3.3 illustrates the situation.)

Claim II: If $q \in Q$ is strictly between u and v, then $q \in Closest(x)$.

Proof of Claim II: Suppose $q \notin Closest(x)$. There exists $z \in D$, $z \neq x$, such that $||q, z|| \leq ||q, x||$.

By the choice of u, we know that $||q, y|| > ||q, x||$; thus $z \neq y$. Thus, $||q, z|| < ||q, y||$. By the choice of y, $||e, y|| \leq ||e, z||$. By Lemma 3.3(a), $||q, y|| \leq ||q, z||$, contradiction. Thus, $q \in Closest(x)$.

To improve the readability of the remaining calculations, we define $a = ||x, v||$, $b = ||e, v||$, $c = ||y, v||$, and $d = ||u, v||$.

$||u, x|| \leq ||u, v|| + ||v, x||$ by the triangle inequality. By definition, $||u, y|| \leq ||u, x||$, and thus, $||u, y|| \leq ||u, v|| + ||v, x||$, that is, $c - d \leq d + a$, and thus $2d \geq c - a$. By Claim II and the definition of u, every node in Q strictly between v and u is closer to x than to any other member of D. Moreover, by Claim I, every node in P is closer to x than to any other member of D. Thus, the cardinality of $Closest_D(x)$ is at least $a + b + d$.

Finally, $2|Closest_D(x)| \geq 2a + 2b + 2d \geq a + 2b + c \geq b + c = ||y, e|| \geq k + 1$. □

Lemma 3.5 *If D is a k-exclusive set in a connected graph of n nodes, then*

$$|D| \leq \max\left\{1, n/\left\lceil \frac{k+1}{2} \right\rceil\right\}$$

Proof: If $|D| \leq 1$, the result is trivial. Otherwise, let $D = \{x_1, \ldots x_m\}$, for $m \geq 2$. Then $\sum_{i=1}^{m} |Closest_D(x_i)| \leq n$, since $Closest_D(x_i) \cap Closest_D(x_j) = \emptyset$ for any $i \neq j$. By Lemma 3.4, $|Closest_D(x_i)| \geq \lceil \frac{k+1}{2} \rceil$ for each i. Thus, $m \leq n/\lceil \frac{k+1}{2} \rceil$. □

Figure 3.4 shows examples of networks with minimal k-dominating sets of maximum size for (a) $k = 4$, $n = 9$, and (b) $k = 5$, $n = 10$. Nodes of the k-dominating sets are indicated by solid dots.

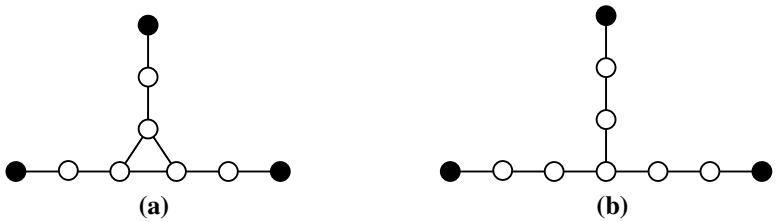


Figure 3.4: Illustration of Lemma 3.5, where $n = 9$ and $k = 4$ for (a), and $n = 10$ and $k = 5$ for (b).

Corollary 3.6 *If D is a solution to the generalized minimal k-dominating set problem, then $|D \setminus R^*| \leq \max\{1, n/\lceil \frac{k+1}{2} \rceil\}$, where n is the number of nodes of G.*

Proof: $D \setminus R^*$ is k-exclusive by Lemma 3.2. The result follows from Lemma 3.5. □

4 The Up to M-Nearest Member Problem

Given a network G, where each process has an identifier, a set Z of processes of G, and a fixed input M, the *up to M-nearest member* problem is to compute the ID of the nearest process of Z to each process, as well as the distance to that nearest process, provided that distance does not exceed M. Since this problem (or a simplified version in which only the distance is computed) is solved several times in this paper, we give a self-stabilizing silent algorithm $\mathcal{Z}$ for the problem. $\mathcal{Z}$ also finds the distance to the second up to M-nearest member of Z, again provided that distance does not exceed M.

For given Z and M, we define the following values for each process x.

1. $DIST\_Z(x) = \min\{M+1, ||x, Z||\}$
2. $NEAREST\_Z(x) = \begin{cases} \perp \text{ if } DIST\_Z(x) = M+1 \\ \max\{y.id : y \in Z \wedge ||x,y|| = DIST\_Z(x)\} \text{ otherwise} \end{cases}$
3. $2^{\text{nd}}DIST\_Z(x) = \min\{M+1, ||x, Z||_2\}$

$\mathcal{Z}$ has three variables:

4. $x.dist\_Z \in \{0 \ldots M+1\}$
5. $x.nearest\_Z$, ID type or $\perp$.
6. $x.2^{\text{nd}}dist\_Z \in \{1 \ldots M+1\}$

and five functions:

7. $Dist\_Z(x) = \begin{cases} 0 \text{ if } x \in Z \\ \min\{M+1, 1+\min\{y.dist\_Z : y \in N(x)\}\} \text{ otherwise} \end{cases}$
8. $Nearest\_Z(x) = \begin{cases} x.id \text{ if } x.dist\_Z = 0 \\ \perp \text{ if } x.dist\_Z = M+1 \\ \max\{y.nearest\_Z : 1 + y.dist\_Z = x.dist\_Z\} \text{ otherwise} \end{cases}$
9. $2^{\text{nd}}Dist\_Z.1(x) = \min\begin{cases} M+1 \\ 1+\min\{y.2^{\text{nd}}dist\_Z : y \in N(x)\} \end{cases}$
10. $2^{\text{nd}}Dist\_Z.2(x) = \min\begin{cases} M+1 \\ 1+\min\{y.dist\_Z : y \in N(x) \wedge y.nearest\_Z \neq x.nearest\_Z\} \end{cases}$
11. $2^{\text{nd}}Dist\_Z(x) = \min\{2^{\text{nd}}Dist\_Z.1(x), 2^{\text{nd}}Dist\_Z.2(x)\}$

The action of $\mathcal{Z}$ is given in Algorithm 4.1.

Algorithm 4.1: $\mathcal{Z}(x)$

```
1: if x.dist_Z ≠ Dist_Z(x) then
2:     x.dist_Z ← Dist_Z(x)
3: end if
4: if x.nearest_Z ≠ Nearest_Z(x) then
5:     x.nearest_Z ← Nearest_Z(x)
6: end if
7: if x.2nd dist_Z ≠ 2nd Dist_Z(x) then
8:     x.2nd dist_Z ← 2nd Dist_Z(x)
9: end if
```

Below, we will show that $\mathcal{Z}$ is silent and *self-stabilizes* to the predicate defined as follows: for every process x,

- $x.nearest\_Z = NEAREST\_Z(x)$, *i.e.*, $x.nearest\_Z$ is the ID of the nearest member of Z to x, provided that the distance between x and that process does not exceed M,
- $x.dist\_Z = DIST\_Z(x)$, *i.e.*, the distance between x and y such that $y.id = x.nearest\_Z$, provided that distance does not exceed M, and

- $x.2^{\text{nd}}dist\_Z = 2^{\text{nd}}DIST\_Z(x)$, *i.e.*, the distance between x and its second up to M-nearest member of Z, provided that distance does not exceed M.

Hence, we define its legitimate configuration as follows.

Definition 4.1 *A configuration of $\mathcal{Z}$ is called* legitimate *if $x.nearest\_Z = NEAREST\_Z(x)$, $x.dist\_Z = DIST\_Z(x)$, and $x.2^{\text{nd}}dist\_Z = 2^{\text{nd}}DIST\_Z(x)$ for every process x.*

Our design of $\mathcal{Z}$ is inspired by Lemma 4.2 below.

Lemma 4.2 *For $Z \subseteq G$ and $x \in G$*
(a) *$DIST\_Z(x) \leq 2^{\text{nd}}DIST\_Z(x)$,*
(b) *$DIST\_Z(y) \leq DIST\_Z(x) + 1$ and $2^{\text{nd}}DIST\_Z(y) \leq 2^{\text{nd}}DIST\_Z(x) + 1$ for any $y \in N(x)$,*
(c) *If $y \in N(x)$ and $NEAREST\_Z(y) \neq NEAREST\_Z(x)$, then $2^{\text{nd}}DIST\_Z(x) \leq 1 + DIST\_Z(y)$.*

Proof:
(a) If $Z = \emptyset$, then $DIST\_Z(x) = 2^{\text{nd}}DIST\_Z(x) = M + 1$, and we are done. Otherwise, as G is connected, $DIST\_Z(x) = \min\{M + 1, ||x, Z||\} \neq \infty$. Let $y \in Z$ such that $||x, y|| = ||x, Z||$. $2^{\text{nd}}DIST\_Z(x) = \min\{M + 1, ||x, Z||_2\} = \min\{M + 1, ||x, Z \setminus \{y\}||\}$. As $Z \setminus \{y\} \subset Z$, $DIST\_Z(x) \leq 2^{\text{nd}}DIST\_Z(x)$.
(b) We first show that $DIST\_Z(y) \leq DIST\_Z(x) + 1$. If $DIST\_Z(x) = M + 1$, then as, by definition, $DIST\_Z(y) \leq M + 1$, we are done. Otherwise, $Z \neq \emptyset$ and there is a path $x, \ldots, z$ from x to some process z of Z of length $DIST\_Z(x) \leq M$. Since x and y are neighbors, there is a path $y, x, \ldots, z$ from y to z of length $DIST\_Z(x) + 1 \leq M + 1$. So, $DISTZ(y) \leq DISTZ(x) + 1$.
We now show that $2^{\text{nd}}DIST\_Z(y) \leq 2^{\text{nd}}DIST\_Z(x) + 1$. If $2^{\text{nd}}DIST\_Z(x) = M + 1$, then as, by definition, $2^{\text{nd}}DIST\_Z(y) \leq M + 1$, we are done. Otherwise, $|Z| \geq 2$ and
- there is a path $x, \ldots, z$ from x to some process z of Z of length $DIST\_Z(x) \leq M$ and
- there is a path $x, \ldots, z'$ from x to some process z' of $Z \setminus \{z\}$ of length $2^{\text{nd}}DIST\_Z(x)$, where $DIST\_Z(x) \leq 2^{\text{nd}}DIST\_Z(x) \leq M$.

Since x and y are neighbors,
- there is a path $y, x, \ldots, z$ from y to z of length $DIST\_Z(x) + 1 \leq M + 1$ and
- there is a path $y, x, \ldots, z'$ from y to z' of length $2^{\text{nd}}DIST\_Z(x) + 1 \leq M + 1$.

Hence, $2^{\text{nd}}DIST\_Z(y) \leq 2^{\text{nd}}DIST\_Z(x) + 1$, and we are done.
(c) If $DIST\_Z(y) \in \{M, M + 1\}$, then as, by definition, $2^{\text{nd}}DIST\_Z(x) \leq M + 1$, we are done. Otherwise, $DIST\_Z(y) \leq M - 1$ and $NEAREST\_Z(y) \neq \perp$. Let $z \in Z$ such that $z.id = NEAREST\_Z(y)$. Since x and y are neighbors, there is a path x, y, ..., $z \in Z$ of length $1 + DIST\_Z(y) \leq M$ and, consequently $NEAREST\_Z(x) \neq \perp$. Let $w \in Z$ such that $w.id = NEAREST\_Z(x)$. By hypothesis, $NEAREST\_Z(x) \neq NEAREST\_Z(y)$, so $w \neq z$. Now, $2^{\text{nd}}DIST\_Z(x) = \min\{M + 1, ||x, Z||_2\} = \min\{M + 1, ||x, Z \setminus \{w\}||\}$. Moreover, since $w \neq z$, $z \in Z \setminus \{w\}$. Thus $2^{\text{nd}}DIST\_Z(x) \leq 1 + DIST\_Z(y)$, and we are done.

□

Lemma 4.3 *Any final configuration of $\mathcal{Z}$ is legitimate.*

Proof: Consider a final configuration of $\mathcal{Z}$.
Claim I: $x.dist\_Z = DIST\_Z(x)$ and $x.nearest\_Z = NEAREST\_Z(x)$ for all x.
Proof of Claim I: We first show by induction on $||x, Z||$ that for all x, if $||x, Z|| = i$, then $x.dist\_Z = \min\{M + 1, i\}$, and if $||x, Z|| > i$, then $x.dist\_Z \geq \min\{M + 1, i + 1\}$.
If $||x, Z|| = 0$, then $x \in Z$ and $x.dist\_Z = 0$. If $||x, Z|| > 0$, then $x \notin Z$, so $x.dist\_Z \geq 1$.
If $||x, Z|| = i + 1$, then, every neighbor y of x satisfies $||y, Z|| \geq i$ and by induction hypothesis, $y.dist\_Z \geq \min\{M + 1, i\}$. Moreover, there is a neighbor z of x such that $||y, Z|| = i$ and, by induction hypothesis, $z.dist\_Z = \min\{M + 1, i\}$. Hence, $\min\{y.dist\_Z : y \in N(x)\} = \min\{M + 1, i\}$ and so, $x.dist\_Z = \min\{M + 1, i + 1\}$.
Thus, for every x, $x.dist\_Z = \min\{M + 1, ||x, Z||\} = DIST\_Z(x)$.

We now show that $x.nearest\_Z = NEAREST\_Z(x)$ for all x by induction on $||x, Z||$. If $x.dist\_Z = 0$, then $DIST\_Z(x) = 0$ too and $\max\{y.id : y \in Z \wedge ||x, y|| = 0)\} = x.id$. Hence, $x.nearest\_Z = x.id = NEAREST\_Z(x)$. If $x.dist\_Z = M + 1$, then $x.nearest\_Z = \perp = NEAREST\_Z(x)$. Finally, if $0 < x.dist\_Z < M + 1$, by induction hypothesis and since $z.dist\_Z = DIST\_Z(z)$ for all z, $\max\{y.nearest\_Z : 1+y.dist\_Z = x.dist\_Z\} = \max\{NEAREST\_Z(y) : 1+||y, Z|| = ||x, Z||\} = NEAREST\_Z(x)$.

Claim II: $x.2^{\text{nd}}dist\_Z = 2^{\text{nd}}DIST\_Z(x)$ for all x.

Proof of Claim II: By induction on $2^{\text{nd}}DIST\_Z(x)$. The case $2^{\text{nd}}DIST\_Z(x) = 0$ is vacuous. Suppose $2^{\text{nd}}DIST\_Z(x) > 0$.

We first prove that $x.2^{\text{nd}}dist\_Z \geq 2^{\text{nd}}DIST\_Z(x)$. If $x.2^{\text{nd}}dist\_Z = M + 1$, we are done. Let $x.2^{\text{nd}}dist\_Z \leq M$. If $x.2^{\text{nd}}dist\_Z = 2^{\text{nd}}Dist\_Z.1(x)$, then, for some $y \in N(x)$,

$$x.2^{\text{nd}}dist\_Z = 1 + y.2^{\text{nd}}dist\_Z = 1 + 2^{\text{nd}}DIST\_Z(y) \geq 2^{\text{nd}}DIST\_Z(x)$$

by the inductive hypothesis and Lemma 4.2. On the other hand, if $x.2^{\text{nd}}dist\_Z = 2^{\text{nd}}Dist\_Z.2(x)$, then for some $y \in N(x)$, $NEAREST\_Z(y) \neq NEAREST\_Z(x) \wedge \ x.2^{\text{nd}}dist\_Z = 1 + dist\_Z(y) = 1 + DIST\_Z(y) \geq DIST\_Z(x)$ by Claim I and Lemma 4.2. Thus both processes of identifiers $NEAREST\_Z(x)$ and $NEAREST\_Z(y)$ lie within distance $x.2^{\text{nd}}dist\_Z$ of x, hence $2^{\text{nd}}DIST\_Z(x) \leq x.2^{\text{nd}}dist\_Z$.

We now prove that $x.2^{\text{nd}}dist\_Z \leq 2^{\text{nd}}DIST\_Z(x)$. If $2^{\text{nd}}DIST\_Z(x) = M + 1$, we are done. Assume $2^{\text{nd}}DIST\_Z(x) \leq M$. Let z be the process such that $z.id = NEAREST\_Z(x)$. Let v be the process such that $v.id = \max\{w.id : w \in Z \setminus \{z\} \wedge ||x, v|| = 2^{\text{nd}}DIST\_Z(x)\}$. Pick $y \in N(x)$ on a shortest path from x to v.

Case 1: $||y, z|| \leq 2^{\text{nd}}DIST\_Z(x) - 1$. Then, $2^{\text{nd}}DIST\_Z(y) = 2^{\text{nd}}DIST\_Z(x) - 1$, hence

$$x.2^{\text{nd}}dist\_Z \leq 2^{\text{nd}}Dist\_Z.1(x) \leq 1 + y.2^{\text{nd}}dist\_Z \leq 2^{\text{nd}}DIST\_Z(x)$$

Case 2: $||y, z|| \geq 2^{\text{nd}}DIST\_Z(x)$. Then $NEAREST\_Z(y) = v.id$, hence

$$x.2^{\text{nd}}dist\_Z \leq 2^{\text{nd}}Dist\_Z.2(x) \leq 1 + y.dist\_Z = 1 + DIST\_Z(y) = 2^{\text{nd}}DIST\_Z(x)$$

The lemma follows from Claims I and II. □

Lemma 4.4 *$\mathcal{Z}$ is self-stabilizing and silent under the distributed unfair daemon.*

Proof: By Lemma 4.3, We need only prove that any computation of $\mathcal{Z}$ is finite.

Let $\Gamma = \gamma_0, \gamma_1 \ldots$ be any computation of $\mathcal{Z}$, and assume Γ is infinite. Let A_1 be the set of processes which execute Line 2 infinitely often, and A_2 be the set of processes which execute Line 5 infinitely often, A_3 be the set of processes which execute Line 8 infinitely often. By Remark 2.1, we can assume, without loss of generality, that processes not in A_1 never execute Line 2, that processes not in A_2 never execute Line 5, and that processes not in A_3 never execute Line 8.

Suppose $A_1 \neq \emptyset$. For all x, let $\lambda(x) = \liminf_{t\to\infty} x.dist\_Z^t$ be the *limit inferior* of the sequence $x.dist\_Z^0, x.dist\_Z^1, \ldots$ Notice that, for all x, the domain of $x.dist\_Z$ is $\{0, ..., M + 1\}$ (where M is fixed), hence we have $\lambda(x) \in \{0, ..., M + 1\}$. Pick a process $x \in A_1$ so that $\lambda(x) = d$ is minimum. If $x \in Z$, then $x.dist\_Z$ is set to zero the first time x executes Line 2, and cannot change at a later step, contradiction. If $d = M+1$, then $x.dist\_Z$ can never exceed d, and thus $x.dist\_Z = M+1$ in every configuration, contradiction. Finally, suppose $0 < d \leq M$. There must be some $y \in N(x)$ such that $y.dist\_Z = d - 1$ infinitely often, and thus $\lambda(y) \leq d - 1$. Because of our choice of x, we know that $y \notin A_1$, and hence $y.dist\_Z = d - 1$ in every configuration of Γ. This implies that $Dist\_Z(x) \leq d$ in every configuration, which implies that $x.dist\_Z \leq d$ in every configuration. By Remark 2.1, without loss of generality, since $\lambda(x) = d$, we have $x.dist\_Z = d$ in every configuration, contradiction.

We now assume that $A_1 = \emptyset$, and $A_2 \neq \emptyset$. Thus, $x.dist\_Z$ is constant for all x. Pick a process $x \in A_2$ so that $x.dist\_Z = d$ is minimum. If $d = 0$, then $x.nearest\_Z$ is set to $x.id$ the first time x executes, and cannot change at a later step, contradiction. If $d = M + 1$, then $x.nearest\_Z$ is set to $\perp$ the first time x executes, and never changes thereafter, contradiction.

Suppose $0 < d \leq M$. Let $Y = \{y \in N(x) : y.dist\_Z = d-1\}$. By hypothesis, $y.nearest\_Z$ never changes for any $y \in Y$, and thus $Nearest\_Z(x)$ is fixed; the first time x executes, the value of $x.nearest\_Z$ will be assigned, and cannot subsequently change, contradiction.

We now assume that $A_1 = A_2 = \emptyset$, and $A_3 \neq \emptyset$. We can prove that this situation is impossible by induction on $2^{\text{nd}}DIST\_Z(x)$, in a manner similar to the previous proofs. We omit the details. □

Lemma 4.5 *$\mathcal{Z}$ converges to a final legitimate configuration within $O(M)$ rounds under the distributed unfair daemon.*

Proof: Pick a process x, and let $d = DIST\_Z(x)$ and $s = 2^{\text{nd}}DIST\_Z(x)$. We also denote that $R(i)$ the index of the last configuration of the ith round in the computation (*n.b.*, from the previous lemma, there is no infinite round).

Claim I: If at least $t \geq 0$ rounds of the computation have elapsed, then (a) $x.dist\_Z = \min\{d, M+1\}$ if $t > d$ and (b) $x.dist\_Z \geq \min\{M+1, t\}$ if $t \leq d$.

Proof of Claim I: By induction on t. If $t = 0$, then (a) is vacuous, and (b) is trivial. If $t > 0$ and $d = 0$, then (b) is vacuous, while (a) is trivial, since $x \in Z$ and x must have executed Line 2 at least once if necessary.

Let $t > 0$ and $d > 0$. By the inductive hypothesis, (a) and (b) hold after $t-1$ rounds have elapsed.

Case 1. $t \leq d$. Then $y.dist\_Z \geq \min\{M+1, d-1\}$ for all $y \in N(x)$ after $t-1$ rounds have elapsed. By the inductive hypothesis, and x must have executed Line 5 at least once if necessary since then, and therefore (b) holds; (a) is vacuous.

Case 2. $t > d$. Pick $z \in N(x)$ such that $DIST\_Z(z) = d-1$. By the inductive hypothesis, after $t-1$ rounds have elapsed, $z.dist\_Z = d-1$ and $y.dist\_Z \geq \min\{M+1, d-1\}$ for all $y \in N(x)$. Thus $x.dist\_Z = d$ after t rounds have elapsed. We have proven (a), and (b) is vacuous. This concludes the proof of Claim I.

Claim II: If at least $d+2$ rounds have elapsed, $x.nearest\_Z = NEAREST\_Z(x)$.

Proof of Claim II: By induction on d. If $d = 0$, then $x \in Z$. Within one round, $x.dist\_Z = 0$, hence $Nearest\_Z(x) = x.id$. Within one more round, $x.nearest\_Z = x.id$.

Suppose $0 < d \leq M$. Let $Y = \{y \in N(x) : ||y, Z|| = d-1\}$. Then, $NEAREST\_Z(x) = \max\{NEAREST\_Z(y) : y \in Y\}$. If at least $d+1$ rounds have elapsed, by Claim I(a), $y.dist\_Z = ||z, Z||$ for all $z \in N(x)$, and by the inductive hypothesis, $y.nearest\_Z = NEAREST\_Z(y)$ for all $y \in Y$; thus $Nearest\_Z(x) = NEAREST\_Z(x)$. Within one more round, we are done.

Finally, consider the case $d = M+1$. If at least $M+2$ rounds have elapsed, by the inductive hypothesis, $x.dist\_Z = M$. Within one more round, $Nearest\_Z(x) = \perp$. This completes the proof of Claim II.

Claim III: If at least t rounds have elapsed, $x.2^{\text{nd}}dist\_Z \geq \min\{t-3, s\}$.

Proof of Claim III: By induction on t. If $t \leq 4$, the claim is trivial, since $s \geq 1$. Let $t > 4$.

Subclaim 1: for all i such that $R(t-1) \leq i \leq R(t)$, we have $2^{\text{nd}}Dist\_Z.1^i(x) \geq \min\{t-3, s\}$.

If $2^{\text{nd}}Dist\_Z.1^i(x) = M+1$, then since $s \leq M$, we are done. Otherwise, there is $y \in N(x)$ such that $2^{\text{nd}}Dist\_Z.1^i(x) = 1 + y.2^{\text{nd}}dist\_Z^i \geq 1 + \min\{t-4, 2^{\text{nd}}DIST\_Z(y)\} \geq \min\{t-3, s\}$ by the inductive hypothesis and Lemma 4.2, and we are done.

Subclaim 2: for all i, $R(t-1) \leq i \leq R(t)$, $2^{\text{nd}}Dist\_Z.2^i(x) \geq \min\{t-3, s\}$.

If $2^{\text{nd}}Dist\_Z.2^i(x) = M+1$, then since $s \leq M$, we are done.

Otherwise, $2^{\text{nd}}Dist\_Z.2^i(x) = 1 + y.dist\_Z^i$ for some $y \in N(x)$, where $y.nearest\_Z^i \neq x.nearest\_Z^i$. By Claim I, $y.dist\_Z^i \geq \min\{t-1, DIST\_Z(y)\}$.

Case 2(a): $t \geq d+4$. Then $t-1 \geq d+3 \geq DIST\_Z(y)+2$. By Claim I, $y.dist\_Z = DIST\_Z(y)$, and by Claim II, $NEAREST\_Z(x) = x.nearest\_Z^i$ and $NEAREST\_Z(y) = y.nearest\_Z^i$, and thus $s \leq 1 + DIST\_Z(y)$. Hence, $2^{\text{nd}}Dist\_Z.2^i(x) \geq \min\{t, s\}$.

Case 2(b): $t \leq d+3$. Then $2^{\text{nd}}Dist\_Z.2^i(x) \geq \min\{t, DIST\_Z(y)+1\} \geq \min\{t, d\} \geq t-3 = \min\{t-3, s\}$.

By Subclaims 1 and 2, if $x.2^{\text{nd}}dist\_Z < \min\{t-3, s\}$ at the beginning of round t, then Line 8 is executed

during the round t so that $x.2^{\text{nd}}dist\_Z$ becomes greater than or equal to $\min\{t-3, s\}$ forever, and we are done.

Claim IV: $x.2^{\text{nd}}dist\_Z = s$ if at least t rounds have elapsed, for $t \geq s+3$.

Proof of Claim IV: By induction on s. The case $s = 0$ is vacuous. Let $s \geq 1$. By Claim III, $2^{\text{nd}}dist\_Z(x) \geq s$, and thus we need only show that $2^{\text{nd}}dist\_Z(x) \leq s$. First, if $s = M+1$, we are done since $x.2^{\text{nd}}dist\_Z \in \{1 \ldots M+1\}$. Hence, assume $s \leq M$ and consider the following two possible cases.

Case 1: $s = 1 + 2^{\text{nd}}DIST\_Z(y)$ for some $y \in N(x)$.

By induction hypothesis, for all $i \geq R(t-1)$, we have $y.2^{\text{nd}}dist\_Z^{\,i} = 2^{\text{nd}}DIST\_Z(y)$. So, $2^{\text{nd}}Dist\_Z^{\,i}(x) \leq 2^{\text{nd}}DIST\_Z(y) + 1$. Hence, within at most one additional round, *i.e.* at last at the end of round t, we have $x.2^{\text{nd}}dist\_Z \leq 1 + 2^{\text{nd}}DIST\_Z(y) = s$ forever (see Lines 7-8), and we are done.

Case 2: $s = 1 + DIST\_Z(y)$ for some $y \in N(x)$ such that $NEAREST\_Z(y) \neq NEAREST\_Z(x)$.

By Claim II, for all $i \geq R(t-1)$, we have $NEAREST\_Z(x) = x.nearest\_Z^{\,i}$ and $NEAREST\_Z(y) = y.nearest\_Z^{\,i}$. So, $2^{\text{nd}}Dist\_Z^{\,i}(x) \leq 1 + y.dist\_Z^{\,i} = 1 + DIST\_Z(y)$ by Claim I. Hence, $2^{\text{nd}}Dist\_Z^{\,i}(x) \leq s$. Again, within at most one additional round, *i.e.* at last at the end of round t, we have $x.2^{\text{nd}}dist\_Z \leq s$ forever (see Lines 7-8), and we are done.

The statement of the lemma follows from Claims I, II, and IV. □

5 Proof Labeling Schemes

Korman *et al.* [14] introduced *proof labeling schemes*. The purpose of such a scheme is to reduce the problem of verifying a global predicate on a network to the problem of verifying a local predicate at each process of the network. We present a modification of their technique.

Korman *et al.* gave examples of proof labeling schemes for some distributed problems, in which computation of a proof labeling, which proves a given configuration is legitimate, is easier, or believed to be easier, than finding a legitimate configuration.

One such example is the coloring problem. It is quite easy to verify that a k-coloring of a graph is correct, but k-colorability of a graph is $\mathcal{NP}$-complete if $k \geq 3$.

5.1 General Definition of Proof Labeling Schemes

Let $\mathbb{P}$ be a global predicate on configurations of a network. A *proof labeling scheme* for $\mathbb{P}$ consists of the following.

1. A variable $x.label$, the *proof label*, or simply *label*, of x, for each process x. A *proof labeling* consists of a proof label per process, *i.e.*, the set $\{x.label : x \in G\}$ is called a proof labeling.
2. A *marker* algorithm Marker, which computes a proof labeling.
3. A *decoder* algorithm Decoder. For any process x, and any proof labeling L, Decoder(x, L) is Boolean, and depends only on the state of x, and on the proof labels of x and its neighbors, such that
   (a) If $\mathbb{P}$ = TRUE for the current configuration of G, then Decoder(x, L) = TRUE for every process x.
   (b) If $\mathbb{P}$ = FALSE for the current configuration of G, and L is any proof labeling, then there is some process x of G for which Decoder(x, L) = FALSE.

   We write Decoder(x) if L is understood.

5.2 $\mathcal{L}$: A Proof Labeling Scheme for the GMDS Problem

We now define a proof labeling scheme $\mathcal{L} = (\mathcal{L}\_\text{Marker}, \mathcal{L}\_\text{Decoder})$ for the generalized minimal k-dominating set problem. $\mathcal{L}$_Marker executes independently of our computation. In fact, $\mathcal{L}$ will work properly with any algorithm for the problem, if the network has unique identifiers.

Let D be any set of processes which satisfies the relative k-dominance and inclusion conditions, namely $N_k(D) = N_k(D^*)$ and $R^* \subseteq D \subseteq D^*$. The correctness predicate is that D is minimal subject to the relative k-dominance and inclusion conditions.

Given any k and any set of processes $D \subseteq G$, we define the following quantities for any process x.[2]

1. $DIST\_D^*(x) = \min\{||x, D^*||, k+1\}$
2. $DIST\_D(x) = \min\{||x, D||, k+1\}$
3. $NEAREST\_D(x) = \begin{cases} \perp \text{ if } DIST\_D(x) = k+1 \\ \max\{y \in D : ||x,y|| = ||x,D||\} \text{ otherwise} \end{cases}$
4. $2^{\text{nd}}DIST\_D(x) = \min\{||x, D||_2, k+1\}$
5. $E = \{x : 2^{\text{nd}}DIST\_D(x) = k+1\}$
6. $DIST\_E(x) = \min\{||x, E||, k+1\}$

Lemma 5.1 *For $D \subseteq G$ and $x \in D$, $Excl\_Followers_{k,D}(x) \subseteq N_k(x) \cap E$.*

Proof: Let $y \in Excl\_Followers_{k,D}(x)$. By definition, $y \in N_k(x)$ and $N_k(y) \cap D = \{x\}$. So, $||y, D||_2 = ||y, D \setminus \{x\}|| > k$. Hence, $2^{\text{nd}}DIST\_D(y) = \min\{||y, D||_2, k+1\} = k+1$ and, so $y \in E$. Thus, $y \in N_k(x) \cap E$, and we are done. □

Lemma 5.2 *Given an input instance (G, k, R^*, D^*) of the GMDS, D is a solution of that instance if and only if the following hold:*

(i) $R^* \subseteq D \subseteq D^*$,
(ii) $(DIST\_D(x) \leq k) \vee (DIST\_D^*(x) = k+1)$ *for all x,*
(iii) $DIST\_E(x) \leq k$ *for all $x \in D \setminus R^*$.*

Proof:

If: Suppose D is a solution to the GMDS problem.

- From Lemma 3.2, we immediately have (i).
- We now prove (ii). Let x be any process. If $x \in N_k(D)$, then $DIST\_D(x) \leq k$, and we are done. Assume $x \notin N_k(D)$. By Lemma 3.2(ii), $x \notin N_k(D^*)$, hence $DIST\_D^*(x) = k+1$, and we are done.
- We now prove (iii). Let $x \in D \setminus R^*$. By Lemma 3.2(iii), we can pick $z \in Excl\_Followers(x)$. By definition, $z \in N_k(x)$, and $z \in E$ by Lemma 5.1. Thus $DIST\_E(x) \leq k$.

Only if: Conversely, suppose conditions (i), (ii), and (iii) hold. It is sufficient to show that Conditions (i)–(iii) in the statement of Lemma 3.2 hold.

- Condition (i) and Lemma 3.2(i) are identical, so Lemma 3.2(i) trivially holds.
- Since $D \subseteq D^*$, we have $N_k(D) \subseteq N_k(D^*)$. Pick $x \in N_k(D^*)$. By definition, $DIST\_D^*(x) \leq k$. By (ii), $DIST\_D(x) \leq k$, *i.e.,* $x \in N_k(D)$. Thus $N_k(D^*) \subseteq N_k(D)$. Thus Lemma 3.2(ii) holds.
- Pick $x \in D \setminus R^*$. Then $DIST\_E(x) \leq k$ by (iii). Pick $z \in E \cap N_k(x)$. Then $z \in Excl\_Followers_{k,D}(x)$, hence Lemma 3.2(iii) holds.

□

Formal Definition of $\mathcal{L}$

[2]These quantities will be used to implement $\mathcal{L}$_Marker as several particular instances of the up to M-nearest member algorithm $\mathcal{Z}$ given in Section 4, see Algorithm 5.1.

Algorithm 5.1: $\mathcal{L}$_Marker(x)

```
1: if x.dist_D* ≠ Dist_D*(x) then
2:     x.dist_D* ← Dist_D*(x)
3: end if
4: if x.dist_D ≠ Dist_D(x) then
5:     x.dist_D ← Dist_D(x)
6: end if
7: if x.nearest_D ≠ Nearest_D(x) then
8:     x.nearest_D ← Nearest_D(x)
9: end if
10: if x.2nd dist_D ≠ 2nd Dist_D(x) then
11:     x.2nd dist_D ← 2nd Dist_D(x)
12: end if
13: if x.dist_E ≠ Dist_E(x) then
14:     x.dist_E ← Dist_E(x)
15: end if
```

We now define the proof labeling scheme $\mathcal{L} = (\mathcal{L}\text{\_Marker}, \mathcal{L}\text{\_Decoder})$ for the GMDS problem. For each process x, the proof label of x consists of the following five variables at each process x.

1. $x.dist\_D^*$, integer in the range $\{0 \dots k+1\}$,
2. $x.dist\_D$, integer in the range $\{0 \dots k+1\}$,
3. $x.nearest\_D$, ID type or $\perp$,
4. $x.2^{\text{nd}}dist\_D$, integer in the range $\{1 \dots k+1\}$,
5. $x.dist\_E$, integer in the range $\{0 \dots k+1\}$.

as well as the corresponding functions:

6. $Dist\_D^*(x) = \begin{cases} 0 \text{ if } x \in D^* \\ \min\{k+1, 1+\min\{y.dist\_D^* : y \in N(x)\}\} \text{ otherwise} \end{cases}$
7. $Dist\_D(x) = \begin{cases} 0 \text{ if } x \in D \\ \min\{k+1, 1+\min\{y.dist\_D : y \in N(x)\}\} \text{ otherwise} \end{cases}$
8. $Nearest\_D(x) = \begin{cases} x.id \text{ if } x.dist\_D = 0 \\ \perp \text{ if } x.dist\_D = k+1 \\ \max\{y.nearest\_D : (y \in N(x)) \wedge (1 + y.dist\_D = x.dist\_D)\} \text{ otherwise} \end{cases}$
9. $2^{\text{nd}}Dist\_D.1(x) = \min \begin{cases} k+1 \\ 1+\min\{y.2^{\text{nd}}dist\_D : y \in N(x)\} \end{cases}$
10. $2^{\text{nd}}Dist\_D.2(x) = \min \begin{cases} k+1 \\ 1+\min\{y.dist\_D : \{y \in N(x)\} \wedge \{y.nearest\_D \neq x.nearest\_D\}\} \end{cases}$
11. $2^{\text{nd}}Dist\_D(x) = \min\{2^{\text{nd}}Dist\_D.1(x), 2^{\text{nd}}Dist\_D.2(x)\}$
12. $Dist\_E(x) = \begin{cases} 0 \text{ if } x.2^{\text{nd}}dist\_D = k+1 \\ \min\{k+1, 1+\min\{y.dist\_E : y \in N(x)\}\} \text{ otherwise} \end{cases}$

$\mathcal{L}$_Marker(x) can be described using one action, which computes the value of each of these constants, storing that value as the correspondingly named proof label variable.

13. $\mathcal{L}\_Enabled(x) \equiv (x.dist\_D^* \neq Dist\_D^*(x)) \vee (x.dist\_D \neq Dist\_D(x)) \vee$
    $(x.nearest\_D \neq Nearest\_D(x)) \vee$
    $\big(x.2^{\text{nd}}dist\_D \neq 2^{\text{nd}}Dist\_D(x)\big) \vee (x.dist\_E \neq Dist\_E(x))$

We define:

14. $\mathcal{L}$_Decoder(x) $\equiv \neg\mathcal{L}\_Enabled(x) \wedge ((x.dist\_D \leq k) \vee (x.dist\_D^* = k+1)) \wedge$
    $((x \in R^*) \vee (x.dist\_E \leq k) \vee (x \notin D))$

By definition of $\mathcal{L}$_Decoder(x), we immediately deduce the following corollary from Lemma 5.2.

Corollary 5.3 *Assume $R^* \subseteq D \subseteq D^*$. $\mathcal{L}$_Decoder$(x) =$ TRUE for all x if and only if D is a solution to the GMDS problem.*

5.2.1 Projection to $\mathcal{Z}$

We analyze $\mathcal{L}$_Marker using the up to M-nearest member algorithm $\mathcal{Z}$ given in Section 4. Let $\mathcal{Z}'$ be the same as $\mathcal{Z}$, but without the variables that relate to up to M-nearest members and secondary distances, and where the action consists of simply the first three lines of Algorithm 4.1. Trivially, Lemmas 4.3, 4.4, and 4.5 apply to $\mathcal{Z}'$ as well. Let Π_1 be the projection of $\mathcal{L}$_Marker to $\mathcal{Z}'$ obtained by mapping D^* to Z, k to M, and $x.dist\_D^*$ to $x.dist\_Z$. Let Π_2 be the projection of $\mathcal{L}$_Marker to $\mathcal{Z}$ obtained by mapping D to Z, k to M, $x.dist\_D$ to $x.dist\_Z$, $x.nearest\_D$ to $x.nearest\_Z$, and $x.2^{\text{nd}}dist\_D$ to $x.2^{\text{nd}}dist\_Z$ for all x.

Theorem 5.4 *Assume $R^* \subseteq D \subseteq D^*$. ($\mathcal{L}$_Marker, $\mathcal{L}$_Decoder) is a proof labeling scheme for the GMDS problem, and $\mathcal{L}$_Marker converges in $O(k)$ rounds.*

Proof: Let Γ be a computation of $\mathcal{L}$_Marker. By Lemmas 4.3, 4.4, and 4.5, using the projections Π_1 and Π_2, within $O(k)$ rounds Γ will reach a configuration $\gamma*$ at which the values of $x.dist\_D^*$, $x.dist\_D$, $x.nearest\_D$, and $x.2^{\text{nd}}dist\_D$ will equal $DIST\_D^*(x)$, $DIST\_D(x)$, $NEAREST\_D(x)$, and $2^{\text{nd}}DIST\_D(x)$, respectively.

Let Γ' be the suffix of Γ which starts at $\gamma*$. Then the set E is fixed throughout Γ'. Let Π_3 be the projection of Γ' to $\mathcal{Z}'$ obtained by mapping E to Z, k to M, and $x.dist\_E$ to $x.dist\_Z$ for all x. Again, by Lemmas 4.3, 4.4, and 4.5, using the projection Π_3, we have that Γ' (and hence Γ) reaches a legitimate final configuration of $\mathcal{L}$_Marker, at which $\mathcal{L}$_Decoder$(x) =$ TRUE for all x if and only if D is a solution to the GMDS problem (*cf.* Remark 5.3), within $O(k)$ additional rounds. Our result follows. □

6 Abstract Algorithms for the GMDS Problem

Before giving our algorithm GMD, we first give two abstract algorithms for the generalized minimal k-dominating set problem. The first of these algorithms, ABST, is given in Section 6.2, assumes global scope, and computes sets R and D at each step. After $O(n/k)$ steps, $R = D$ is the solution to the problem. The second abstract algorithm, SYNC, is given in Section 6.3 and implements each step of ABST as one iteration of a loop, referred to as the *main loop* of SYNC in the following. Note that the main loop consists of $2k+3$ steps. SYNC is essentially a distributed algorithm using a synchronous daemon, but we again assume global scope. In Section 7, we give GMD as an self-stabilizing implementation of SYNC in the asynchronous composite model.

6.1 Sets R^*, R, D, and D^*

Our three algorithms have as inputs two sets R^* and D^* of nodes of a connected graph such that $R^* \subseteq D^*$. They repeatedly compute two other sets of nodes, R and D. In the three algorithms, we require that the *inclusion condition* $R^* \subseteq R \subseteq D \subseteq D^*$ holds in any configuration; other than that restriction, R and D are initialized arbitrarily.

Membership in the sets R^*, R, D, and D^* is encoded with an integer variable $rank(x) \in \{1,2,3,4,5\}$, *the rank of a process* x, where

$$\begin{aligned} R^* &= \{x : x.rank = 1\} \\ R &= \{x : x.rank \leq 2\} \\ D &= \{x : x.rank \leq 3\} \\ D^* &= \{x : x.rank \leq 4\} \end{aligned}$$

Hence, $G \setminus D^* = \{x : x.rank = 5\}$.

The relationship between ranks and the sets R^*, R, D, and D^* is illustrated in Figures 6.1 and 6.2. As we can see, the encoding inherently guarantees the following desirable property, regardless of the values of the *rank* variables.

Remark 6.1 *The inclusion condition $R^* \subseteq R \subseteq D \subseteq D^*$ holds in any configuration.*

Then, we should emphasize that the membership in sets R^* and D^* must not be modified by the three algorithms. In order to do so, we proceed as follows in the code of our algorithms.

- Any node x of rank 1 or 5 cannot modify its rank.
- No node can set its rank to 1 or 5.

Finally, notice that, for the sake of simplicity, *rank* is implicit in the code of Algorithms ABST and SYNC.

6.2 The Abstract Algorithm ABST

The code of ABST is given in Algorithm 6.3. We recall that $R^* \subseteq D^*$ are two input sets of nodes of a connected graph. The algorithm ABST repeatedly computes two sets of nodes, R and D, until $R = D$. D will be then the solution of the GMDS problem. We define the *main loop invariant* of ABST to be the conjunction of the following conditions.

1. D is *k-dominating relative to D^**, meaning that $N_k(D) = N_k(D^*)$.
2. $\mathit{Excl\_Followers}_{k,R}(x) \neq \emptyset$ for all $x \in R \setminus R^*$.

Initially, since D and R are arbitrary (except that they satisfy the inclusion condition $R^* \subseteq R \subseteq D \subseteq D^*$), the main loop invariant may not hold. However, it will hold after *at most one* iteration. At each subsequent iteration of the main loop of ABST, nodes can be added to D and nodes can be deleted from R, and the loop invariant will be maintained. The computation ends when $R = D$; at that time, the main loop invariant guarantees that D satisfies the output conditions.

In the definitions below, k, R^*, and D^* are understood.

1. We define a new type, called the *augmented* ID type: a value of that type is either $\perp$ ("undefined"), or the ordered pair $x.\widetilde{id} = (\mathit{rank}(x), x.id)$ (the *augmented identifier* of x). We take $\perp$ be the minimum value of this type; otherwise, we use the lexical order. If S is a non-empty set of processes, we define $\min S$ and $\max S$ to be the process in S of minimum, respectively maximum, augmented identifier. Thus $\max \emptyset = \perp$.
2. $\mathit{Leader}_{D,R}(x) = \min \{y \in N_k(x)\}$.
3. $\mathit{Followers}_{D,R}(x) = \{y : \mathit{Leader}_{D,R}(y) = x\}$
4. $\mathit{Excl\_Followers}_D(x) = \{y : N_k(y) \cap D = \{x\}\}$. Note that the set $\mathit{Excl\_Followers}_D(x)$ is identical to the set $\mathit{Excl\_Followers}_{k,D}(x)$ defined in Section 3.1.

Lemma 6.2 *$\mathit{Excl\_Followers}_D(x) \subseteq \mathit{Followers}_{D,R}(x)$ for any $x \in D$.*

Proof: Let $y \in \mathit{Excl\_Followers}_D(x)$. x is the unique element of $N_k(y) \cap D$. So, for every $z \in N_k(y) \setminus \{x\}$, $z.\widetilde{id} > x.\widetilde{id}$. Hence, $\mathit{Leader}_{D,R}(y) = x$ and so, $y \in \mathit{Followers}_{D,R}(x)$. □

Intuition During execution of ABST, the main loop invariant is first established, and then maintained. The first iteration is exceptional: if the main loop invariant does not hold initially, nodes can be added or deleted from both D and R during that iteration. During subsequent iterations, nodes are neither added to D nor deleted from R. In each iteration of the algorithm, any node of D that has no followers is removed from D, since it is not needed in the k-dominating set relative to D^*, while any node not in R that has at least one exclusive follower is moved to R. Figure 6.1 illustrates the movement of these nodes during any one iteration (other than the first one) using an Euler diagram where the universe is the set of all nodes. The computation ends when $D = R$, as illustrated in Figure 6.2.

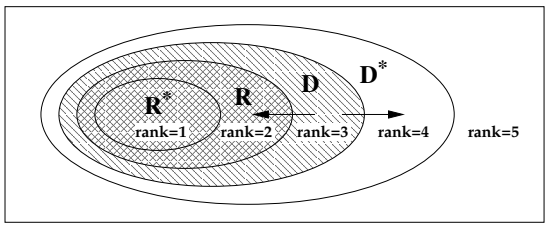


Figure 6.1: After the 1st iteration of ABST, nodes move from $D \setminus R$ to $D^* \setminus D$ or $R \setminus R^*$.

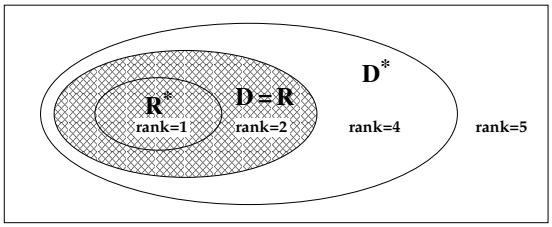


Figure 6.2: When ABST is done, $D = R$ will be a solution.

Recall that the code of ABST is given in Algorithm 6.3. In the table, D^t and R^t denote the sets D and R after t iterations of the loop; in particular, D^0, R^0 are the initial "arbitrary" sets. We write $\mathit{Followers}^t = \mathit{Followers}_{D^{t-1},R^{t-1}}$, $\mathit{Excl\_Followers}^t = \mathit{Excl\_Followers}_{D^{t-1}}$, and $\mathit{Leader}^t(x) = \mathit{Leader}_{D^{t-1},R^{t-1}}(x)$.

Algorithm 6.3: Code of ABST

1: $t \leftarrow 0$
2: **repeat**
3: $\quad t \leftarrow t + 1$
4: $\quad R^t \leftarrow R^* \cup \{x \in D^{t-1} : \mathit{Excl\_Followers}^t(x) \neq \emptyset\}$
5: $\quad D^t \leftarrow R^* \cup \{x \in D^* : \mathit{Followers}^t(x) \neq \emptyset\}$
6: **until** $R^t = D^t$
7: **output** D^t

6.2.1 Analysis of ABST

The main result of this section is that ABST converges in $O(n/k)$ iterations of its loop. Note that the case $t = 0$ is excluded from the statements of the remarks and lemmas given below. First, note that, from Lemma 6.2 and the code of Algorithm 6.3, sets R^t and D^t with $t \geq 1$ are well-defined *w.r.t.* Remark 6.1. Hence, we reformulate Remark 6.1 as follows.

Remark 6.3 *$R^* \subseteq R^t \subseteq D^t \subseteq D^*$ if $t \geq 1$.*

From Remark 6.3 and the code of Algorithm 6.3, follows.

Remark 6.4 *$R^t \subseteq D^{t-1}$ if $t \geq 1$.*

Lemma 6.5 *D^t is a k-dominating set relative to D^* if $t \geq 1$.*

Proof: We need to prove that every process $x \in N_k(D^*)$ lies in $N_k(y)$ for some $y \in D^t$. Let $y = \mathit{Leader}^t(x)$, which is in D^* since $x \in N_k(D^*)$. Then $y \in D^t$, since $x \in \mathit{Followers}^t(y)$. □

Lemma 6.6 *$D^{t+1} \subseteq D^t$ if $t \geq 1$.*

Proof: Let $x \in D^{t+1}$. By definition, $x \in D^*$ and $\mathit{Followers}^{t+1}(x) = \mathit{Followers}_{D^t,R^t}(x) \neq \emptyset$. Let $y \in \mathit{Followers}_{D^t,R^t}(x)$. Then, $\mathit{Leader}_{D^t,R^t}(y) = x$. In particular, $y \in N_k(x)$ and so $y \in N_k(D^*)$. Now, D^t is k-dominating relative to D^* by Lemma 6.5, so $y \in N_k(D^t)$, and then $x = \mathit{Leader}_{D^t,R^t}(y) = \min\{z \in N_k(y)\} \in D^t$. □

By Lemma 6.6 and the definition of *Excl_Followers*, we have the following corollary.

Corollary 6.7 *Excl_Followers*$^{t+2}(x) \supseteq$ *Excl_Followers*$^{t+1}(x)$ *if* $x \in D^{t+1}$ *and* $t \geq 1$.

Lemma 6.8 $R^t \supseteq R^{t-1}$ *if* $t \geq 3$.

Proof: Let $x \in R^{t-1}$. If $x \in R^*$, then we are done. Otherwise, $x \in D^{t-1}$ (by Remark 6.3) and *Excl_Followers*$^t(x) \supseteq$ *Excl_Followers*$^{t-1}(x) \neq \emptyset$ by Corollary 6.7, hence $x \in R^t$. □

Lemma 6.9 *If* $t \geq 2$, *then* $R^{t+1} = D^{t+1}$ *or* $R^t \neq R^{t+2}$.

Proof: If $R^t \neq R^{t+1}$, we are done by Lemma 6.8. Suppose $R^t = R^{t+1} \neq D^{t+1}$. By Remark 6.3, $R^{t+1} \subsetneq D^{t+1}$. Pick $x = \max\left(D^{t+1} \setminus R^{t+1}\right) = \max\left(D^{t+1} \setminus R^t\right)$. Pick $y \in$ *Followers*$^{t+1}(x)$ (by definition of x, *Followers*$^{t+1}(x) \neq \emptyset$). Then, by definition of *Followers*, we have $x = \min\left(N_k(y) \cap D^t\right)$ and $N_k(y) \cap R^t = \emptyset$. By Lemma 6.6, $N_k(y) \cap D^{t+1} \subseteq N_k(y) \cap D^t$, and thus $x = \min\left(N_k(y) \cap D^{t+1}\right)$. Conversely, $x = \max\left(N_k(y) \cap D^{t+1}\right)$, since $N_k(y) \cap D^{t+1} = N_k(y) \cap \left(D^{t+1} \setminus R^t\right) \subseteq D^{t+1} \setminus R^t$. Thus x is the only element of $N_k(y) \cap D^{t+1}$, *i.e.,* $y \in$ *Excl_Followers*$^{t+2}(x)$. Hence, $x \in R^{t+2} \setminus R^t$. □

Lemma 6.10 ABST *computes a solution to the GMDS problem.*

Proof: By Lemmas 6.6, 6.8, 6.9, and Remark 6.3, ABST must halt. Let T be the number of iterations of the loop. $R^T = D^T$ is k-dominating relative to D^* by Lemma 6.5. We need to show the minimality of D^T. Suppose $\widetilde{D}$ is k-dominating relative to D^* for some $R^* \subseteq \widetilde{D} \subsetneq D^T$. Pick $x \in D^T \setminus \tilde{D}$. Since $x \in R^T \setminus R^*$, there exists some $y \in$ *Excl_Followers*$^T(x) =$ *Excl_Followers*$_{D^{T-1}}(x) \subseteq N_k(D^{T-1})$ and so, $y \in N_k(D^*)$, by Remark 6.3. Then $\tilde{D} \cap N_k(y) \subsetneq D^{T-1} \cap N_k(y) = \{x\}$, hence $\tilde{D} \cap N_k(y) = \emptyset$, contradicting the hypothesis that $\tilde{D}$ is k-dominating relative to D^*. □

Lemma 6.11 *The number of iterations of the loop of the Abstract Algorithm does not exceed*

$$\max\left\{5, \tfrac{2n}{\lceil \frac{k+1}{2} \rceil} + 3\right\} = O\left(\tfrac{n}{k}\right)$$

Proof: Let T be the number of iterations of the loop. If $T \leq 3$, we are done. Suppose $T \geq 4$.

For any $x \in R^T \setminus R^*$, *Excl_Followers*$_{D^T}(x) =$ *Excl_Followers*$^{T+1}(x) \supseteq$ *Excl_Followers*$^T(x) \neq \emptyset$ by Corollary 6.7. Then $R^T \setminus R^*$ is k-exclusive since $D^T = R^T$.

$R^{t-1} \subseteq R^t$ for all $3 \leq t \leq T$, by Lemma 6.8. Let $\Delta^t = R^t \setminus R^{t-1}$. By Remark 6.3, $R^T \setminus R^*$ is the disjoint union

$$R^T \setminus R^* = \left(R^2 \setminus R^*\right) \cup \Delta^3 \cup \Delta^4 \cup \cdots \cup \Delta^{T-1} \cup \Delta^T$$

By Lemma 6.9, $\Delta^{t-1} \cup \Delta^t \neq \emptyset$ for all $4 \leq t \leq T$. Thus, the cardinality of $\Delta^3 \cup \Delta^4 \cup \cdots \cup \Delta^{T-1} \cup \Delta^T$ is at least $\lfloor (T-2)/2 \rfloor$. The cardinality of $R^T \setminus R^*$ is at least that large, and thus

$$\tfrac{T-3}{2} \leq \lfloor \tfrac{T-2}{2} \rfloor \leq \left|R^T \setminus R^*\right| \leq \max\left\{1, n/\lceil \tfrac{k+1}{2} \rceil\right\}$$

by Lemma 3.5, since $R^T \setminus R^*$ is k-exclusive. The result follows. □

In our implementation of ABST by GMD in Section 7, ABST may be required to iterate after it has already converged, now we have:

Lemma 6.12 *If the loop of* ABST *iterates after* $R = D$, *the sets* R *and* D *do not change.*

Proof: Let T be the number of iterations of the loop. Assume, by contradiction, that $R^t = D^t = R^T = D^T$ does not hold for some $t > T$. Without loss of generality, assume t is minimum. By definition of T, $R^T = D^T$, so either $R^t \neq D^T$, or $D^t \neq D^T$.

If $R^t \neq D^T$, then, by minimality of t, $R^t \neq D^{t-1}$. By Remark 6.4, $R^t \subsetneq D^{t-1}$, so there exists $x \in D^{t-1} \setminus R^* = D^T \setminus R^*$ such that *Excl_Followers*$^t(x) =$ *Excl_Followers*$_{D^{t-1}}(x) =$ *Excl_Followers*$_{D^T}(x) = \emptyset$ meaning that D^T is not a solution to the GMDS problem (Lemma 3.2), a contradiction to Lemma 6.10. Hence, $R^t = D^T$.

Now, $R^t = D^T$ implies that $D^t = D^T$, since $D^t \subseteq D^{t-1} = D^T$ (Lemma 6.6) and $D^T = R^t \subseteq D^t$ (Remark 6.3). Hence, $R^t = D^t = R^T = D^T$ holds, a contradiction. □

6.3 The Synchronous Algorithm SYNC

In Algorithm 6.4 below, we give SYNC, the *synchronous array algorithm*, a slightly lower level implementation of ABST, although still abstract. Each iteration of the loop of ABST is implemented as one iteration of lines 2 through 19 of Algorithm 6.4, which we call the *main loop* of SYNC. Each process x has four array variables which, are recalculated during each iteration of the main loop:

1. $x.min\_\widetilde{id}[\ ]$, of augmented identifier type,
2. $x.max\_\widetilde{id}[\ ]$, of augmented identifier type,
3. $x.max\_min\_\widetilde{id}[\ ]$, of augmented identifier type,
4. $x.exclusive[\ ]$, of Boolean type.

Each array has length $k+1$, indexed by $\{0 \dots k\}$.

Lemma 6.13 *At Line 15 of* SYNC*:*

(a) *For* $0 \le i \le k$, $x.min\_\widetilde{id}[i] = min\Big\{y.\widetilde{id} : y \in N_i(x)\Big\}$.

(b) *For* $0 \le i \le k$, $x.max\_\widetilde{id}[i] = max\Big\{y.\widetilde{id} : y \in N_i(x) \cap D\Big\}$.

(c) *For* $0 \le j \le k$, $x.max\_min\_\widetilde{id}[j] = max\Big\{y.min\_\widetilde{id}[k] : y \in N_j(x)\Big\}$.

(d) *For* $0 \le j \le k$, $x.exclusive[j] \iff \Big(\exists y \in N_j(x) : y.min\_\widetilde{id}[k] = y.max\_\widetilde{id}[k]\Big)$.

(e) $x.max\_min\_\widetilde{id}[k] = x.\widetilde{id} \iff \Big(\exists y \in N_k(x) : y.min\_\widetilde{id}[k] = x.\widetilde{id}\Big)$.

(f) $(x \in D \wedge x.exclusive[k]) \iff (\exists y : N_k(y) \cap D = \{x\})$.

Proof: We prove (a) and (b) by induction on i. If $i = 0$, (a) holds trivially. If $x \in D$, (b) is trivial; otherwise, we need to use the default rule that $\max \emptyset = \perp$.

If $i > 0$, (a) and (b) follow from the fact that $N_i(x) = \bigcup \{N_{i-1}(y) : y \in N_1(x)\}$.

We prove (c) and (d) by induction on j. For $j = 0$, (c) and (d) both follow from the definitions. For $j > 0$, (c) and (d) follow from the fact that $N_j(x) = \bigcup \{N_{j-1}(y) : y \in N_1(x)\}$.

The "$\Longrightarrow$" part of (e) follows from (c). Conversely, pick $y \in N_k(x)$ where $y.min\_\widetilde{id}[k] = x.\widetilde{id}$. By (c), we have $x.max\_min\_\widetilde{id}[k] \ge y.min\_\widetilde{id}[k] = x.\widetilde{id}$. Pick $z \in N_k(x)$ such that $x.max\_min\_\widetilde{id}[k] = z.min\_\widetilde{id}[k]$. We have $x.max\_min\_\widetilde{id}[k] = z.min\_\widetilde{id}[k] \le x.\widetilde{id}$, since $x \in N_k(z)$, and we are done.

We now prove the "$\Longrightarrow$" part of (f). By (d), $\exists y \in N_j(x) : y.min\_\widetilde{id}[k] = y.max\_\widetilde{id}[k]$. Assume, by contradiction, that $N_k(y) \cap D \setminus \{x\} \ne \emptyset$. Let $z \in N_k(y) \cap D \setminus \{x\}$. Without loss of generality, assume that $z.\widetilde{id} < x.\widetilde{id}$. By (a), $y.min\_\widetilde{id}[k] \le z.\widetilde{id} < x.\widetilde{id}$. By (b), $y.max\_\widetilde{id}[k] \ge x.\widetilde{id}$, since $x \in D$. Hence, $y.min\_\widetilde{id}[k] \ne y.max\_\widetilde{id}[k]$, a contradiction.

To prove the converse, pick $y \in N_k(x)$ where $N_k(y) \cap D = \{x\}$. By (a), $y.min\_\widetilde{id}[k] = x.\widetilde{id}$, and by (b), $y.max\_\widetilde{id}[k] = x.\widetilde{id}$. By (d), we are done. □

Algorithm 6.4: Synchronous Algorithm SYNC

1: **repeat**
2: **for** all x in parallel **do**
3: $x.min\_\widetilde{id}[0] \leftarrow x.\widetilde{id}$
4: $x.max\_\widetilde{id}[0] \leftarrow \begin{cases} x.\widetilde{id} \text{ if } x \in D \\ \perp \text{ otherwise} \end{cases}$
5: **for** $1 \leq i \leq k$ **do**
6: $x.min\_\widetilde{id}[i] \leftarrow \min\left\{y.min\_\widetilde{id}[i-1] : y \in N_1(x)\right\}$
7: $x.max\_\widetilde{id}[i] \leftarrow \max\left\{y.max\_\widetilde{id}[i-1] : y \in N_1(x)\right\}$
8: **end for**
9: $x.max\_min\_\widetilde{id}[0] \leftarrow x.min\_\widetilde{id}[k]$
10: $x.exclusive[0] \leftarrow \begin{cases} \text{TRUE if } x.max\_\widetilde{id}[k] = x.min\_\widetilde{id}[k] \\ \text{FALSE otherwise} \end{cases}$
11: **for** $1 \leq j \leq k$ **do**
12: $x.max\_min\_\widetilde{id}[j] \leftarrow \max\left\{y.max\_min\_\widetilde{id}[j-1] : y \in N_1(x)\right\}$
13: $x.exclusive[j] \leftarrow \bigvee \{y.exclusive[j-1] : y \in N_1(x)\}$
14: **end for**
15: {arrays have been computed}
16: $R \leftarrow R^* \cup \{x \in D : x.exclusive[k]\}$
17: $D \leftarrow R^* \cup \left\{x \in D^* : x.max\_min\_\widetilde{id}[k] = x.\widetilde{id}\right\}$
18: {The value of $rank(x)$ is updated for all x}
19: **end for**
20: **until** $R = D$

By claims (e) and (f) of Lemma 6.13, Lines 16 and 17 in Algorithm SYNC respectively compute the same values for sets R and D as Lines 4 and 5 in Algorithm ABST. Hence, the result follows.

Corollary 6.14 *For given input sets $R^* \subseteq R \subseteq D \subseteq D^*$, the sets R and D after t iterations of the main loop of* SYNC *are equal to the sets R^t and D^t after t steps of* ABST.

Lemma 6.15 SYNC *computes a solution to the* GMDS *problem in $O(n)$ iterations of its main loop.*

Proof: By Lemma 6.10, Corollary 6.14, and Lemma 6.11. □

In Section 8, we will be concerned about what happens if we force extra iterations of the main loop of SYNC. From Lemma 6.12, we immediately have:

Remark 6.16 *If the main loop of* SYNC *iterates after D is a solution, then the sets R and D do not change.*

7 The Algorithm GMD

In this section, we define GMD, an asynchronous distributed implementation of SYNC, which is self-stabilizing and silent under the distributed unfair daemon. In the trivial special case that $k > n$, GMD converges in $O(k)$ rounds and the set $D \setminus R^*$ computed by GMD contains at most one process. If $k \leq n$, GMD converges in $O(N)$ rounds, where N is a given upper bound on n.

7.1 Overview of GMD

GMD consists of five *modules*.

Unison Our first module is a unison, $\mathcal{U}$, essentially the unison of Boulinier *et al.* [26]. Such an algorithm implements a local clock at each process, such that once stabilized, every clock increments infinitely often while ensuring that the difference between any two neighboring clocks is always at most one. This algorithm has two input parameters: K and α, see Figure 7.1. K is the period of the clocks and α is the number of states used by a reset mechanism that allows the convergence, *i.e.*, when a process locally detects an error, it resets its clock to $-\alpha$.

Computation of Arrays and Ranks Our second module, $\mathcal{D}$, emulates SYNC, and is the module that actually computes array variables and determines the sets R and D. Once $\mathcal{U}$ stabilizes, $\mathcal{D}$ consists of repeated iterations of a *main loop* during which four arrays of identifiers, each of size $O(k)$, are computed at each process. The execution of each iteration is synchronized by $\mathcal{U}$. The arrays would take $O(k \log n)$ space if all were retained, but the elements of each array are deleted as soon as they are no longer needed, and only $O(\log n)$ space is needed at each process. $\mathcal{D}$ converges to a legitimate configuration within $O(n/k)$ iterations of the main loop.

Proof Labeling Our third module, $\mathcal{L}$, is the *proof labeling scheme* given in Section 5, an ordered pair ($\mathcal{L}$_Marker, $\mathcal{L}$_Decoder). When $\mathcal{L}$_Decoder(x) = TRUE for all x, the set D is a solution to the problem instance. Its memory requirement is $O(\log k + \log n)$.

Up to M-Nearest Member Our fourth module, $\mathcal{Z}$, has two forms. The simple form calculates, for each process x, the minimum distance from x to a given set Z of processes. The general form calculates both the minimum distance from each x to Z and the distance to the second closest member of Z, and the ID of the closest member of Z. This module is used several times by the modules $\mathcal{L}$ and GMD. Its memory requirement is $O(\log k + \log n)$.

The Master Module GMD The master module which we name GMD, the same as the entire algorithm, controls the usage of the other modules, and also has its own variables. Its memory requirement is $O(\log k + \log N)$.

7.2 The Unison $\mathcal{U}$

Algorithm $\mathcal{U}$, the *unison*, adapted from Boulinier *et al.* [26, 27], is the first module of GMD, see Algorithm 7.2 for its formal code. $\mathcal{U}$ works under the distributed unfair daemon and uses k and N as parameters. The unison can be defined independent of subsequent modules. For the sake of completeness, we give the definition of $\mathcal{U}$ here. $\mathcal{U}$ has one variable, eight functions, and three actions. It also uses constants $K = (2k+3)\left\lceil \frac{N+1}{2k+3} \right\rceil$ and $\alpha = N - 2$.

1. $x.r$, integer in the range $-\alpha \ldots K-1$. Thus, the memory requirement of the algorithm is $O(\log k + \log N)$.
2. $\mathit{Near}(a,b) \equiv (0 \leq a, b < K) \wedge ((a = b) \vee (a = (b+1) \bmod K) \vee (b = (a+1) \bmod K))$, Boolean. Note that *Near* is reflexive and symmetric.
3. $\mathit{Unison\_Error}(x) \equiv (0 < x.r < K) \wedge (\exists y \in N(x) : \neg \mathit{Near}(x.r, y.r))$, Boolean.
4. $\mathit{Can\_Converge}(x) \equiv (-\alpha \leq x.r < 0) \wedge (\forall y \in N(x) : (x.r \leq y.r \leq 0))$, Boolean.
5. $\mathcal{U}\_\mathit{Safe}(x) \equiv (\forall y \in N(x) : \mathit{Near}(x.r, y.r))$, Boolean. We say that $\mathcal{U}$ is *safe* if $\mathcal{U}\_\mathit{Safe}(x)$ for all x.
6. $\mathit{Can\_Tick}(x) \equiv \mathcal{U}\_\mathit{Safe}(x) \wedge \Big(\forall y \in N(x) : \Big(y.r \in \{x.r, (x.r+1) \bmod K\}\Big)\Big)$, Boolean.
7. $\mathit{Clock}(x) = x.r \bmod (2k+3)$, integer in the range $0 \ldots 2k+2$.
8. $\mathit{Must\_Tick}(x) \equiv \mathit{Can\_Tick}(x) \wedge ((x.r \neq 0) \vee (\exists y \in N(x) : (y.r = 1)))$, Boolean.
9. $\mathit{Must}\mathcal{U}(x) \equiv \mathit{Must\_Tick}(x) \vee \mathit{Unison\_Error}(x) \vee \mathit{Can\_Converge}(x)$, Boolean.

The functions *Clock*, *Must_Tick*, and *Must*$\mathcal{U}$ are not actually used in Algorithm 7.2 but are part of the interface between $\mathcal{U}$ and later modules.

We use the definitions and theorems of Boulinier *et al.* [26, 27]. The evolution of the value of $x.r$ is as illustrated in Figure 7.1 below. Once safety has been achieved, *i.e.,* $\mathcal{U}\_Safe(x) = \text{TRUE}$ for all x, and the values of $x.r$ are confined to the cycle, *i.e.,* the set of values indicated by solid dots in the figure.

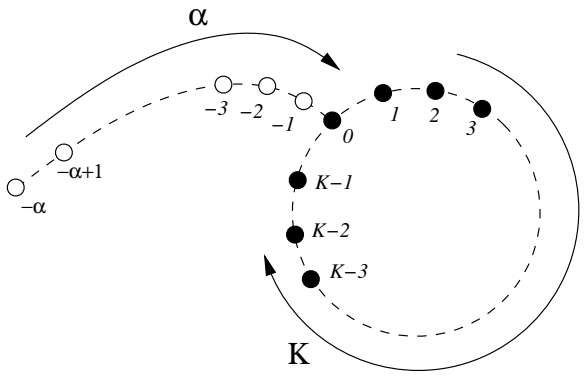


Figure 7.1: Possible values of $x.r$. The upper arrow shows the increment of $x.r$ when x executes a convergence action, while the circular arrow shows increment when x executes a normal action. When x executes a reset action, the value of $x.r$ jumps to $-\alpha$. If the unison is in a safe configuration, all values of $x.r$ are in the cyclic portion of the figure, indicated by solid black dots.

$\mathcal{U}$ is written in [26, 27] as follows:

$$\begin{array}{rrcl} \text{Reset Action} & \mathit{Unison\_Error}(x) & \longrightarrow & x.r \leftarrow -\alpha \\ \text{Convergence Action} & \mathit{Can\_Converge}(x) & \longrightarrow & x.r \leftarrow x.r + 1 \\ \text{Normal Action} & \mathit{Can\_Tick}(x) & \longrightarrow & x.r \leftarrow (x.r + 1) \bmod K \end{array}$$

Adapting the notation of [26, 27] in this paper, we redefine $\mathcal{U}$ in Algorithm 7.2. In the sequel, we also use the predicate $\mathcal{U}\_Enabled(x)$ defined below, for all process x. $\mathcal{U}\_Enabled(x)$ is true if and only if x is enabled in $\mathcal{U}(x)$.

$$\mathcal{U}\_Enabled(x) \equiv \mathit{Unison\_Error}(x) \vee \mathit{Can\_Converge}(x) \vee \mathit{Can\_Tick}(x)$$

Algorithm 7.2: $\mathcal{U}(x)$

```
1: if Unison_Error(x) then
2:     x.r ← −α {reset action}
3: else if Can_Converge(x) then
4:     x.r ← x.r + 1 {convergence action}
5: else if CanTick(x) then
6:     x.r ← (x.r + 1) mod K {normal action}
7: end if
```

Let $Cyclo_G$ be the *cyclomatic characteristic* of G, *i.e.,* the length of the shortest maximal cycle of a cycle basis of G if G is cyclic, and 2 if G is a tree. Let T_G be the length of the longest chordless cycle in a graph G. Choosing $K > Cyclo_G$ and $\alpha \geq T_G - 2$ yields a correct unison [26]. Now, as no elementary cycle in an arbitrary graph G can have length greater than n, $Cyclo_G \leq n \leq N$. Then, if G is acyclic, $T_G = 2$. In any case, $T_G \leq n = N$. Thus, for any (connected) network G, $T_G \leq n \leq N$ and $Cyclo_G \leq n \leq N$, and consequently, the choice of $K = (2k+3)\left\lceil \frac{N+1}{2k+3} \right\rceil$ and $\alpha = N - 2$ yields a correct unison in any connected network, as stated in Theorem 7.1 below.

Theorem 7.1 (Boulinier *et al.* [26]) *Given a computation of $\mathcal{U}$ on a connected network G, during which $K = (2k+3)\left\lceil \frac{N+1}{2k+3} \right\rceil$ and $\alpha = N-2$ are fixed:*

(a) *safety of $\mathcal{U}$ is closed,*
(b) *safety of $\mathcal{U}$ is an attractor[3] of* TRUE*,*
(c) *if $\mathcal{U}$ is not safe, then $Unison\_Error(x) \vee Can\_Converge(x) =$ TRUE for some process x,*
(d) *if $\mathcal{U}$ is safe, then $Can\_Tick(x) =$ TRUE for some process x, and*
(e) *$\mathcal{U}$ will be safe within $O(K+\alpha)$ rounds,* i.e. *$O(N+k)$ rounds.*

Write $r(x) = x.r$. If the unison is safe, then *Clock* is a composition of r with the natural projection $\pi : \mathbb{Z}_K \to \mathbb{Z}_{2k+3}$ (recall that K is a multiple of $2k+3$) and that Figure 7.3 is a commutative diagram.

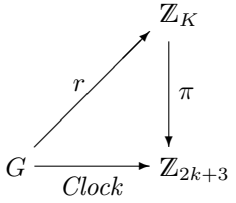


Figure 7.3: Commutative diagram showing r as a lifting of *Clock* to $\mathbb{Z}_K$, where $\mathbb{Z}_m$ denotes the Abelian group of integers modulo m and G is the network.

7.3 The Array Update Module $\mathcal{D}$

It is impossible to consider the actions of the second module, $\mathcal{D}$, independently of $\mathcal{U}$, since every action of $\mathcal{D}$ must be executed simultaneously with the Normal Action of $\mathcal{U}$.

7.3.1 Input sets R^* and D^*

Recall that the membership in the sets R^*, R, D, and D^* is encoded with an integer variable $rank(x) \in \{1,2,3,4,5\}$. This encoding ensures that the *inclusion* condition, $R^* \subseteq R \subseteq D \subseteq D^*$, holds at the (arbitrary) initial configuration as well as throughout the computation (Remark 6.1). Moreover, the membership in the sets R^* and D^* is not modified by GMD. Notice that the knowledge of these inputs need only to be local: any process only needs to know whether or not it belongs to those sets. Finally, the external (*w.r.t.* GMD) application that provides the inputs should be able to modify it, yet in a distributed manner:

- To add a process x to R^*: $rank(x) \leftarrow 1$.
- To add a process x to $D^* \setminus R^*$ (*e.g.*, move a process x from R^* to $D^* \setminus R^*$): $rank(x) \leftarrow v$, where v is any value in $\{2,3,4\}$
- To remove a process x from D^*: $rank(x) \leftarrow 5$.

(To solve the original k-dominating set problem, values 1 and 5 should be simply excluded from *rank* variables.) Of course, our algorithm ABST stabilizes only if no external application modifies its input sets meanwhile. We make this assumption from now on.

7.3.2 Arrays

$\mathcal{D}$ has four array variables for each process, each with indices in the range $\{0 \ldots k\}$. Three of those arrays contain augmented identifiers, and the fourth contains Booleans. The space complexity of the module $\mathcal{D}$ should thus be $O(k \log n)$ per process. However, once any value of any array is no longer needed, it is deleted, saving space. As it turns out, we need retain at most four array values in each process at any give step, following an approach similar to [28]. The space complexity of $\mathcal{D}$ is thus $O(\log n)$.

[3]Predicate P is an attractor for predicate Q if every computation starting from a configuration satisfying Q eventually reaches a configuration satisfying P.

7.3.3 Code of $\mathcal{D}$

For each process x, $\mathcal{D}$ has the following variables and functions.

1. $rank(x) \in \{1, 2, 3, 4, 5\}$, as described in Section 6. Recall that $x.\widetilde{id} = (rank(x), x.id)$, the augmented identifier of x. As before, $rank(x)$ is fixed if its value is either 1 or 5.
2. $x.min\_\widetilde{id}[i]$, for $0 \leq i \leq k$, which will be computed to be the minimum augmented identifier of any process in $N_i(x)$.
3. $x.max\_\widetilde{id}[i]$, for $0 \leq i \leq k$, which will be computed to be the maximum augmented identifier of any process in $N_i(x) \cap D$. If that set is empty, $x.max\_\widetilde{id}[i]$ will be assigned to $\perp$. In the ordering of augmented identifiers, we treat $\perp$ as the minimum element.
4. $x.max\_min\_\widetilde{id}[i]$, for $0 \leq i \leq k$, which will be computed to be the maximum value of $y.min\_\widetilde{id}[k]$ over all $y \in N_i(x)$.
5. $exclusive[i]$, for $0 \leq i \leq k$, Boolean. Let $E = \left\{x : x.min\_\widetilde{id} = x.max\_\widetilde{id}\right\}$. Then $x.exclusive[i]$ will be computed to be TRUE if $N_i(x) \cap E$ is not empty, FALSE otherwise.

$\mathcal{D}$ has only one action, which we call $\mathcal{D}(x)$, and which we define in Algorithm 7.4 below. The interface variable $Clock(x)$ of $\mathcal{U}$ is used to synchronize computation of the arrays.

Algorithm 7.4: $\mathcal{D}(x)$

1: **if** $Clock(x) = 0$ **then**
2: **retain** $x.max\_min\_\widetilde{id}[k]$ and $x.exclusive[k]$
3: **else if** $Clock = i$ for $1 \leq i \leq k+1$ **then**
4: **retain** $x.min\_\widetilde{id}[i-1]$ and $x.max\_\widetilde{id}[i-1]$
5: **else** $\{Clock(x) = i$ for $k+2 \leq i \leq 2k+1\}$
6: **retain** $x.max\_min\_\widetilde{id}[i-k-2]$ and $x.exclusive[i-k-2]$
7: **end if**
8: **delete** all array entries of x not retained {to save space}
9: **if** $Clock(x) = 0$ **then**
10: $x.min\_\widetilde{id}[0] \leftarrow x.\widetilde{id}$
11: **if** $x \in D$ **then**
12: $x.max\_\widetilde{id}[0] \leftarrow x.\widetilde{id}$
13: **else**
14: $x.max\_\widetilde{id}[0] \leftarrow \perp$
15: **end if**
16: **else if** $Clock(x) = i$ for $0 < i \leq k$ **then**
17: $x.min\_\widetilde{id}[i] \leftarrow \min\left\{y.min\_\widetilde{id}[i-1] : y \in N_1(x)\right\}$
18: $x.max\_\widetilde{id}[i] \leftarrow \max\left\{y.max\_\widetilde{id}[i-1] : y \in N_1(x)\right\}$
19: **else if** $Clock(x) = k+1$ **then**
20: $x.max\_min\_\widetilde{id}[0] \leftarrow x.min\_\widetilde{id}[k]$
21: $x.exclusive[0] \leftarrow \left(x.min\_\widetilde{id}[k] = x.max\_\widetilde{id}[k]\right)$
22: **else if** $Clock(x) = i$ for $k+1 < i \leq 2k+1$ **then**
23: $x.max\_min\_\widetilde{id}[i-k-1] \leftarrow \max\left\{y.max\_min\_\widetilde{id}[i-k-2] : y \in N_1(x)\right\}$
24: $x.exclusive[i-k-1] \leftarrow \bigvee\left\{y.exclusive[i-k-2] : y \in N_1(x)\right\}$
25: **end if**
26: **if** $(Clock(x) = 2k+2) \wedge (rank(x) \in \{2,3,4\})$ **then**
27: **if** $x.exclusive[k]$ **then**
28: $rank(x) \leftarrow 2$ {That is, $x \in R \setminus R^*$ after this step}
29: **else if** $x.max\_min\_\widetilde{id}[k] = x.\widetilde{id}$ **then**
30: $rank(x) \leftarrow 3$ {That is, $x \in D \setminus R$ after this step}
31: **else**
32: $rank(x) \leftarrow 4$ {That is, $x \in D^* \setminus D$ after this step}
33: **end if**
34: **end if**

$\mathcal{D}(x)$ is the heart of GMD. Each process x executes that action exactly $2k+3$ times for each emulation of one iteration of the abstract algorithm ABST given in Algorithm 6.3. During that emulation, the value of $Clock(x)$ rotates from 0 to $2k+2$, then back to 0. During each such cycle, the value of every entry in each of the four array variables of x is recomputed. During each execution of $\mathcal{D}(x)$, at most one entry in two of those arrays will be computed; those values will be retained for the next execution of $\mathcal{D}(x)$, and will be deleted during the following execution. Thus, at most four array values are stored at x at once.

7.4 Code of GMD

We list the variables of GMD:

1. All the variables of $\mathcal{D}$, $\mathcal{U}$, and $\mathcal{L}$.
2. $x.f \geq 0$, non-negative integer, the *finishing level*, the current estimate of the distance to the nearest process that detects that computation of $\mathcal{D}$ is not finished.

and functions:

3. All the functions of $\mathcal{D}$, $\mathcal{U}$, and $\mathcal{L}$.
4. *Locally_finshd*$(x) = (rank(x) \neq 3) \wedge \mathcal{L}$_Decoder$(x)$. Note that if $rank(x) = 3$, then $x \in D \setminus R$.
5. $F(x) = \begin{cases} 0 \text{ if } \neg \textit{Locally\_finshd}(x) \\ \min\{1 + 2K,\ 1 + x.f,\ 1 + \min\{y.f : y \in N(x)\}\} \text{ otherwise} \end{cases}$
6. *Finished*$(x) \equiv x.f = 2K + 1$, Boolean.

GMD has one action, GMD(x), defined in Algorithm 7.5. That action is enabled for x if execution of the code would cause some variable of x to be changed.

In the code below, the only line that actually executes the emulation of SYNC is Line 7; the other lines are needed to keep track of the unison and the proof labeling system.

Algorithm 7.5: GMD(x)

```
1: if Must_U(x) ∨ (U_Enabled(x) ∧ ¬ Finished(x)) then
2:    if Unison_Error(x) then
3:       x.r ← −α {reset action of unison}
4:    else if Can_Converge(x) then
5:       x.r ← x.r + 1 {convergence action of unison}
6:    else if Can_Tick(x) ∧ (Must_Tick(x) ∨ ¬Finished(x)) then
7:       D(x) {compute array values}
8:       x.r ← (x.r + 1) mod K {normal action of unison}
9:    end if
10: end if
11: if x.f ≠ F(x) then
12:    x.f ← F(x)
13: end if
14: if L_Enabled(x) then
15:    L_Marker(x) {update proof label}
16: end if
```

7.5 Legitimate Configurations

We now define legitimacy for GMD. Clearly, we need legitimacy to imply that D is a solution to the given instance of the GMDS problem, as defined in Section 3.3. However, we need the legitimacy predicate to be closed, and thus we attach additional conditions. We say a configuration of GMD is *legitimate* if the following four conditions hold:

1. $x.r = 0$ for all x.
2. *Finished*(x) for all x.
3. $\mathcal{L}$_Decoder(x) is true for all x, which implies that D is a solution to the problem.
4. $rank(x) \neq 3$ for all x, *i.e.,* $R = D$.

We remark that the unison is safe in a legitimate configuration.

8 Correctness and Complexity of GMD

The following theorem is proven using Theorems 8.6 and 8.18, below.

Theorem 8.1 GMD *is self-stabilizing and silent under the distributed unfair daemon, and reaches a legitimate configuration within $O(N + k)$ rounds from an arbitrary initial configuration, where N is a given upper bound on n (the number of nodes).*

8.1 Path Delay

In the unison algorithm, clocks are periodic modulo K. Consequently, the values of that variable cannot be used directly to compare them. We illustrate our argument below.

- In a configuration, where the only values are 0 and 1, 0 is the smallest value meaning that every process whose clock has value 0 is late.
- But, in a configuration, where the only values are $K-1$ and 0, $K-1$ is the smallest value meaning that every process whose clock has value $K-1$ is late.

In order to reason about the clock values in a safe configuration, we need to be able to compare the clocks of arbitrarily far processes. Given two process x and y, we must decide whether $x.r$ is late, equal, or ahead compared to $y.r$. For that purpose, the notion of *Path delay* (*delay* in short) is defined in [26, 27]. The path delay from x to y is noted $y \ominus x$. Let $d = y \ominus x$.

- If $d < 0$, then $x.r$ is $-d$ increments ahead of $y.r$.
- If $d = 0$, then $x.r = y.r$.
- If $d > 0$, then $x.r$ is d increments late of $y.r$.

The notion of delay is first defined between neighboring processes as follows. If $y \in N(x)$ and $Near(x.r, y.r)$, let $y \ominus x = \begin{cases} -1 \text{ if } (y.r+1) \bmod K = x.r \\ 0 \text{ if } y.r = x.r \\ 1 \text{ if } y.r = (x.r+1) \bmod K \end{cases}$

The delay is generalized to paths as follows. First, $x \ominus x = 0$. Then, for every two distinct processes x and y, the *path delay* from x to y is $y \ominus x = \sum_{i=0}^{m-1} x_{i+1} \ominus x_i$, where $x = x_0, x_1, \ldots x_m = y$ is any path from x to y in G.

We define a configuration of GMD to be *safe* if the projected configuration of $\mathcal{U}$ is safe. That is, if $Near(x.r, y.r)$ holds for all x and all $y \in N(x)$.

Remark 8.2 *Safety is a closed predicate.*

Lemma 8.3 *If $\mathcal{U}$ is in a safe configuration, the value of $x \ominus y$ is well-defined,* i.e., *does not depend on the choice of path from x to y.*

Proof: From [26]. □

In the remark below we recall some properties given in [26].

Remark 8.4 *If $\mathcal{U}$ is in a safe configuration, and x, y, z are processes, then*

(a) $x \ominus x = 0$,
(b) $y \ominus x = -x \ominus y$,
(c) $x \ominus y + y \ominus z = x \ominus z$,
(d) *If $x.r$ increments and $y.r$ does not increment in a given step, then $x \ominus y$ increases by 1 in that step.*
(e) *If $x.r$ increments and $y.r$ does not increment in a given step, then $y \ominus x$ decreases by 1 in that step.*
(f) *If $x.r$ and $y.r$ both increment in a given step, then $x \ominus y$ does not change in that step.*

We illustrate the notion of delay in Figure 8.1, where increments are modulo $K = 8$. In this figure, the delay from a to b, *i.e.,* $b \ominus a$, is equal to 1, meaning that $a.r$ is one increment late of $b.r$. On the contrary, $a \ominus b = -1$: $b.r$ is one increment ahead of $a.r$. $d \ominus a = 2$. Consider, for example, the path a, b, c, d: $d \ominus a = b \ominus a + c \ominus b + d \ominus c = 1 + 1 + 0 = 2$. But, we could consider another path like a, b, c, h, d, since in a safe configuration the delay is not dependent on the choice of the path from a to b: $d \ominus a = b \ominus a + c \ominus b + h \ominus c + d \ominus h = 1 + 1 + 1 - 1 = 2$. Notice also that $a \ominus d = -d \ominus a = -2$. Consider, for example, the path d, c, b, a: $a \ominus d = c \ominus d + b \ominus c + a \ominus b = 0 + -1 + -1 = -2$. Finally, $d \ominus a = d \ominus f + f \ominus a$, since $d \ominus f = 2$ and $f \ominus a = 0$.

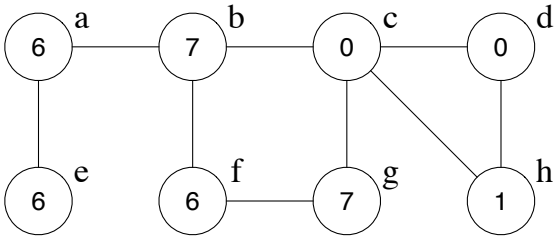


Figure 8.1: Safe configuration ($K = 8$). Clock values are inside nodes.

Lemma 8.5 *If $\mathcal{U}$ is safe, then either Must_Tick(x) =* TRUE *for some x, or $x.r = 0$ for all x.*

Proof: Pick an arbitrary process x^*. Let $m = \max_{x \in G} \{x^* \ominus x\}$. Since $x^* \ominus x^* = 0$ (Remark 8.4.(a)), we have $m \geq 0$. Let $M = \{x : x^* \ominus x = m\}$. Note that *Can_Tick*$(x)$ = TRUE for all $x \in M$.

Case 1: $M = G$. If $x^*.r = 0$, then $x.r = 0$ for all x, and we are done. If $x^*.r \neq 0$, then *Must_Tick*(x^*) = TRUE, and we are done.

Case 2: $M \neq G$. Pick $x \in M$ such that $N(x) \not\subseteq M$. Then $y \ominus x = 1$ for some $y \in N(x)$, hence *Must_Tick*(x) = TRUE, and we are done. □

8.2 Self-Stabilization and Silence of GMD

We now prove self-stabilization and silence of GMD. In the remainder of this section, line numbers refer to lines of the code of GMD, as given in Algorithm 7.5, unless otherwise specified.

Theorem 8.6 *From an arbitrary initial configuration, any computation of* GMD *ends in a legitimate final configuration.*

The proof of Theorem 8.6 follows from Lemma 8.10 and Lemmas 8.11-8.17, given below.

Lemma 8.7 *If R and D remain fixed, then* Line 12 *and* Line 15 *are executed only finitely often.*

Proof: By Lemma 4.4, $\mathcal{L}$_Decoder(x) is eventually fixed for all x, and thus Line 15 is executed only finitely many times.

Computation of F is an instance of the nearest member problem, as given in Section 4, where $Z = \{x : \textit{Locally\_finshd}(x) = \text{FALSE}\}$ and each execution of Line 12 is an execution of $\mathcal{Z}$. By Lemma 4.4, there are only finitely many such executions. □

For convenience, we define Block[2:9] to be the block of code from Line 2 to Line 9, inclusive A selected process x executes Block[2:9] if and only if the condition given in Line 1 is true; we refer to this condition as the Block[2:9]-condition.

Lemma 8.8 *In any infinite computation of* GMD*, there are infinitely many executions of* Block[2:9].

Proof: Assume that Γ is an infinite computation of GMD during which there are only finitely many executions of Block[2:9]. By Remark 2.1, without loss of generality, no process executes Block[2:9] during Γ.

Since the variables of $\mathcal{U}$ never change, D and R are fixed. By Lemma 8.7, Line 12 and Line 15 are each executed only finitely often. Thus Γ is finite, contradiction. □

Lemma 8.9 *In any infinite computation of* GMD*, all processes execute* Block[2:9] *infinitely many times.*

Proof: Let Γ be an infinite computation of GMD. Suppose some process executes Block[2:9] only finitely often, hence executes actions of $\mathcal{U}$ only finitely often. By [26, 27], since G contains no chordless cycle of length greater than $\alpha + 1$, no other process executes an action of $\mathcal{U}$ infinitely often, hence the number of executions of Block[2:9] is finite, contradicting Lemma 8.8. □

Lemma 8.10 *Any final configuration of* GMD *is legitimate.*

Proof: Let γ be a final configuration of GMD, *i.e.,* no process is enabled at γ. We need to verify that the four conditions listed in Section 7.5 hold.

Since the Block[2:9]-condition is false for all x, $\mathit{Must}\mathcal{U}(x) = \text{FALSE}$ for all x, which implies that $\mathit{Must\_Tick}(x) = \text{FALSE}$ for all x. Thus, Condition 1 holds, by Lemma 8.5. By Condition 1, $\mathcal{U}\_\mathit{Enabled}(x) = \text{TRUE}$ for all x. Thus $\mathit{Finished}(x) = \text{TRUE}$, since the Block[2:9]-condition is false, *i.e.,* Condition 2 holds.

From Condition 2 and the fact that Line 12 does not execute, we can conclude that $\mathit{Locally\_finshd}(x) = \text{TRUE}$ for all x. This implies Conditions 3 and 4. □

We define the predicate $\mathbb{R}$ to be true if $\mathit{Near}(x.r, y.r)$ for all x and y such that $x.r > 0$ and $y \in N(x)$ (In other words, for all x, either $x.r \leq 0$, or $\mathit{Near}(x.r, y.r)$ for all $y \in N(x)$). When $\mathbb{R}$ holds, we say that the configuration is *reset-free*. Lemma 8.11 below states that $\mathbb{R}$ is closed and an attractor of TRUE, *i.e.,* regardless the initial configuration, every computation converges to a configuration from which $\mathbb{R}$ holds forever.

Lemma 8.11 $\mathbb{R}$ *is closed and is an attractor of* TRUE.

Proof: By [26, 27], $\mathbb{R}$ is closed. Let Γ be a computation of GMD. If Γ is finite, we are done by Lemma 8.10.

Assume Γ is infinite. By Lemma 8.8, every process executes Block[2:9] infinitely often, which implies that Γ consists of infinitely many rounds, by Lemma 8.9. By [26, 27] there can be only finitely many reset actions. During each round, as long as $\mathbb{R}$ does not hold, at least one process executes Line 3. By [26, 27] there can be only finitely many reset actions, hence eventually $\mathbb{R}$ holds. □

We define the predicate $\mathbb{S}$ to be true if the configuration is *safe*. Recall that a configuration is safe if $\mathcal{U}\_\mathit{Safe}(x)$ for all x.

Lemma 8.12 $\mathbb{S}$ *is closed and is an attractor of* $\mathbb{R}$.

Proof: By [26, 27], $\mathbb{S}$ is closed. Let Γ be a computation of GMD in which $\mathbb{R}$ holds in all configurations. If Γ is finite, we are done by Lemma 8.10.

Assume Γ is infinite. By Lemma 8.8, every process executes Block[2:9] infinitely often, which implies that Γ consists of infinitely many rounds, by Lemma 8.9. Let $\mathit{Min\_r} = \min\{x.r\}$ at any given configuration. If $\mathit{Min\_r} < 0$, then $\mathit{Min\_r}$ increases during the next round. When $\mathit{Min\_r} \geq 0$, $\mathbb{S}$ holds. □

We now define a mapping from safe computations of GMD to computations of SYNC.

The Function $\varrho(x)$ Let $\pi' : \mathbb{Z} \to \mathbb{Z}_K$ be the natural projection. Write $r(x) = x.r$. If $\mathbb{S}$ holds, let $\varrho(x)$ be an integral function, which we call a *lifting* of r, such that Figure 8.2 is a commutative diagram, $\varrho(x) - \varrho(y) = x \ominus y$ for all x, y, and $\varrho(x)$ increments each time $x.r$ increments. If γ_s is the first safe configuration of Γ, we require that, at γ_s, $\varrho(x) \leq 0$ for all x, and $\varrho(x) > -K$ for some x. Thus, $-2K + 1 \leq \varrho(x) \leq 0$ for all x at γ_s.

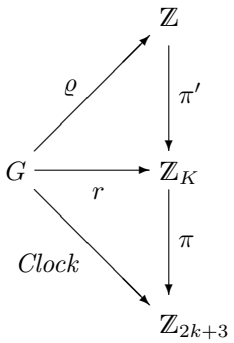


Figure 8.2: Commutative diagram showing ϱ as a lifting of r to the integers, where G is the network.

Angle Notation If $V(x)$ is any variable or function of GMD, we let $V(x)\langle i\rangle$ be the value of $V(x)$ in the first configuration where $\varrho(x) = i$, provided such a configuration exists.

Let Γ be a safe computation of GMD, *i.e.,* a computation starting from a safe configuration. By definition, the safety predicate $ is satisfied in every configuration of Γ. Recall that $x.r$ remains in the set $\{0 \dots K-1\}$, illustrated as the ring in Figure 7.1, and $|x \ominus y| \leq 1$ for any neighboring processes x and y. Our method is to construct a computation $\Lambda = \Lambda(\Gamma) = \lambda_0, \lambda_1 \dots$ of SYNC which computes the same sets R and D as Γ. Since Λ converges to a solution to the GMDS problem, Γ converges to the same solution.

Recall that $-2K + 1 \leq \varrho(x) \leq 0$ for all x in the first configuration of Γ. The main loop of SYNC makes use of only one input variable, namely $rank(x) \in \{1, 2, 3, 4, 5\}$ for each process x. Recall that $x.rank = 1$ and never changes if $x \in R^*$, and that $x.rank = 5$ and never changes if $x \notin D^*$. In our mapping, we set the initial value of $rank(x)$ in $\Lambda(\Gamma)$ equal to the value $rank(x)\langle 0\rangle$ of Γ.

The execution of one iteration of the main loop in Algorithm 6.4 consists of $2k + 3$ steps. The following variables are computed: $x.min\_\widetilde{id}[i]$ for each $0 \leq i \leq k$, during the $(i+1)^{\text{st}}$ step, $x.max\_\widetilde{id}[i]$ for each $0 \leq i \leq k$, during the $(i+1)^{\text{st}}$ step, $x.max\_min\_\widetilde{id}[i]$ for each $0 \leq i \leq k$, during the $(k+i+2)^{\text{nd}}$ step, $x.max\_excl\_\widetilde{id}[i]$ for each $0 \leq i \leq k$, during the $(k+i+2)^{\text{nd}}$ step, and finally, a new value of $rank(x)$ is computed during the $(2k+3)^{\text{rd}}$ step. SYNC converges when there is no process whose *rank* is equal to 3.

Since GMD is asynchronous, there is no one-to-one correspondence between steps of GMD and steps of SYNC. Instead, each step of Λ is emulated in possibly different steps of Γ, but in the same "virtual" step, as enforced by the unison. More specifically, if V is some variable of SYNC and $V^{t,j}$ is the value of $V(x)$ computed at the j^{th} step of the t^{th} iteration of Λ, that same value of $V(x)$ is computed by Γ when $\varrho(x) = (2k+3)t + j$; that is, $V^{t,j}(x) = V(x)\langle(2k+3)t + j\rangle$.

Lemma 8.13 *For any $t \geq 0$ and any process x, the value of $rank(x)$ after t iterations of the main loop of Λ is equal to $rank(x)\langle(2k+3)t\rangle$.*

Proof: The functions and actions of $\mathcal{D}$ exactly mimic the functions and actions of SYNC. In particular, if $\varrho(x) = i = t(2k+3) + j$, then the guard of every action of $\mathcal{D}$ of x is exactly the guard of x after the j^{th} step of the t^{th} iteration of the main loop of SYNC in the computation Λ. □

Let T be the number of iterations of the main loop of ABST. Recall that R^t and D^t are the sets computed by ABST after t iterations of that loop.

Corollary 8.14 *For any $t \geq 0$, let R^t, D^t be the sets computed after t iterations of the main loop of* SYNC *in the computation Λ.*

(a) *If $t \leq T$, then $R\langle t(2k+3)\rangle = R^t$ and $D\langle t(2k+3)\rangle = D^t$.*
(b) *$T = O(n/k)$, $R^T = D^T$, and D^T is a solution to the problem.*
(c) *If $i \geq T(2k+3)$, then $R\langle i\rangle = D\langle i\rangle = D^T$.*

Proof: Part (a) follows immediately from Lemma 8.13, (b) follows from Lemmas 6.10 and 6.11, and Corollary 6.14, and (c) follows from Remark 6.16. □

We define the predicate $\mathbb{V}$ ("$\mathbb{V}$" stands for *verum*, meaning *true*) to be true if $\mathbb{S}$ holds and $\mathcal{D}$ has reached a legitimate configuration. More formally, $\mathbb{V}$ holds in configuration γ if

- $\mathbb{S}$ holds in γ,
- $R = D$ and D is a solution of the generalized minimal k-dominating set problem, and
- both R and D remain fixed in any configuration reachable from γ, *i.e.,* the values of the arrays are such that no further execution of Line 7 will change those sets.

Lemma 8.15 $\mathbb{V}$ *is closed and is an attractor of* $\mathbb{S}$.

Proof: Trivially, $\mathbb{V}$ is closed. Let Γ be a safe configuration. Let γ be the first configuration for which $\varrho(x) \geq T(2k+3)$ for all x. By Corollary 8.14, $\mathbb{V}$ holds at γ. If $\gamma \mapsto \cdots \mapsto \gamma'$, then $\varrho(x) \geq T(2k+3)$ for all x at γ' also, hence $\mathbb{V}$ holds at γ'. Thus, $\mathbb{V}$ is an attractor of $\mathbb{S}$. □

We define the predicate $\mathbb{LV}$ to be true if $\mathbb{V}$ holds, and for all x, $\mathcal{L}$_Decoder$(x) =$ TRUE and $F(x) = 2K+1$. This implies that *Finished*$(x) =$ TRUE for all x.

Lemma 8.16 $\mathbb{LV}$ *is closed and an attractor of* $\mathbb{V}$.

Proof: $\mathbb{LV}$ is closed since there can be no execution of Line 12 or 15 if $\mathbb{LV}$ holds.

Suppose γ is a configuration of a computation Γ, and γ satisfies $\mathbb{V}$. Then, $R = D$ and D is a solution of the problem. In particular, (1) $rank(x) \neq 3$, for all x. Since $\mathbb{V}$ is closed, there will be no further changes in the sets R, D. By Lemma 8.7, Γ contains only finitely many executions of Lines 12 and 15. As soon as there are no more executions of Lines 12 and 15, $\mathcal{L}$_Decoder(x) and *Locally_finshd*(x) are true for ever for all x (by Remark 6.1, Theorem 5.4, and (1)), which implies that $F(x) = 2K+1$ forever. □

We define the predicate $\mathbb{F}$ to be true if the configuration is final. $\mathbb{F}$ is obviously closed. In Lemma 8.16 below, we prove that $\mathbb{F}$ is closed and an attractor of $\mathbb{V}$.

Consider a configuration γ satisfying $\mathbb{LV}$ but not $\mathbb{F}$. From γ, processes execute Lines 7 and 8 until all clocks reach 0. To show Lemma 8.16, we define a potential Φ which actually gives the total number of clock increments required to reach a configuration where all clocks are equal to 0 from any such configuration γ. So, Φ is equal to the sum of local potentials $\phi(x)$, for all x, where $\phi(x)$ gives the number of increments x must perform. Formally, we consider a process x^* whose clock is ahead of all other clocks in γ, *i.e.,* x^* satisfies $x^* \ominus x \geq 0$ for all x, and $\phi(x^*)$ is equal to the number of increments it requires to reach 0. Then, for all x, $\phi(x)$ is equal to $\phi(x^*)$ plus the delay from x to x^* in γ. Notice that, due to the asynchronism, the initial condition on x^* (*i.e.,* the clock of x^* is ahead of others) may not be always satisfied along the execution from γ.

Lemma 8.17 $\mathbb{F}$ *is closed and an attractor of* $\mathbb{LV}$.

Proof: Trivially, $\mathbb{F}$ is closed, since it only holds for a final configuration, by Lemma 8.10.

Suppose a computation $\Gamma = \gamma_0, \gamma_1, \ldots$, such that γ_0 satisfies $\mathbb{LV}$.

We define a potential Φ, which is equal to the number of times the normal action of unison must be executed in order to reach $\mathbb{F}$.

We give the formal definition of Φ. Pick a process x^* such that $x^* \ominus x \geq 0$ for any process x in γ_0. In any configuration of Γ, let $\phi(x^*) = \left\{ \begin{array}{c} 0 \text{ if } x^*.r = 0 \\ K - x^*.r \text{ otherwise} \end{array} \right\}$ a non-negative integer, the number of steps to 0 along the ring shown in Figure 7.1. Then, for any process $x \neq x^*$, let $\phi(x) = \phi(x^*) + x^* \ominus x$. Notice that, since $x^* \ominus x^* = 0$ (Remark 8.4.(a)), we more generally have $\phi(x) = \phi(x^*) + x^* \ominus x$ for all process x. Finally, let $\Phi = \sum \{\phi(x)\}$.

Observation I: $\phi(x) \geq 0$ in γ_0 for any process x, hence $\Phi \geq 0$ in γ_0.

Observation II: If $\phi(x^*) > 0$ and x^* executes Line 8, then $\phi(x)$ is unchanged for any x that does not execute that line in that step, since $\phi(x^*)$ decrements and $x^* \ominus x$ increments, by Remark 8.4.(d).

Observation III: If $\phi(x^*) > 0$ and x^* executes Line 8, then $\phi(x)$ decrements for any x (including x^*) which executes that line at that step, since $\phi(x^*)$ decrements and $x^* \ominus x$ remains unchanged, by Remark 8.4.(f).

Observation IV: For any $x \neq x^*$, if $\phi(x) > 0$ and x executes Line 8, then $\phi(x)$ decrements (whether or not x^* also executes Line 8 in the same step). Indeed, if x^* also executes Line 8, $\phi(x)$ decrements by Observation III. Otherwise, $\phi(x^*)$ remains unchanged, but $x^* \ominus x$ decrements, by Remark 8.4.(e) hence $\phi(x)$ does to.

Claim I: For all any x, if $\phi(x) = 0$, then x cannot execute Line 8.

Proof of Claim I: Assume, by the contradiction, there is a step $\gamma_i \mapsto \gamma_{i+1}$ of Γ where some process x satisfying $\phi(x) = 0$ in γ_i executes Line 8 in $\gamma_i \mapsto \gamma_{i+1}$. Without loss of generality, assume that $\gamma_i \mapsto \gamma_{i+1}$ is the earliest step of Γ subject to that condition. By Observations I-IV, every process z satisfies $\phi(z) \geq 0$ in every configuration of $\gamma_0 \dots \gamma_i$ and $\phi(x) = -1$ in γ_{i+1} (*n.b.*, from γ_0 only Line 8 modifies $z.r$ for all process z). Then, in γ_i, $\phi(x) = 0$ implies that $x.r = 0$. Moreover, x being enabled in γ_i, we have $y.r \in \{0, 1\}$ in γ_i for all process $y \in N(x)$. Assume that there is some process $y' \in N(x)$ such that $y'.r = 1$ in γ_i. Then, $x \ominus y' = -1$ in γ_i. Now, in γ_i $\phi(y') = \phi(x^*) + x^* \ominus y' = \phi(x^*) + x^* \ominus x + x \ominus y' = \phi(x) - 1 = -1$ (*cf.*, Remark 8.4.(c)), a contradiction. So, we have $y.r = 0$ in γ_i for all process $y \in N(x)$ and, so *Must_Tick*(x) = TRUE. Since $\mathbb{LV}$ holds, *Finished*(x) = TRUE. Thus, x cannot execute Line 8, a contradiction.

Observation V: $\Phi \geq 0$ in all configurations of Γ (By Observation I and Claim I).

Claim II: If $\Phi > 0$, there is some process which is enabled to execute Line 8.

Proof of Claim II: Let $Max\_\phi = \max\{\phi(x)\}$.

Case 1: $\phi(x) = Max\_\phi$ for all x. $Max\_\phi > 0$, since otherwise $\Phi = 0$. *Must_Tick*(x) = TRUE for every x, and thus every process is enabled to execute Line 8.

Case 2: Not Case 1. Pick x such that $\phi(x) = Max\_\phi$ and $\phi(y) = Max\_\phi - 1$ for some $y \in N(x)$. Then *Must_Tick*(x) = TRUE, hence x is enabled to execute Line 8.

Claim III: $\Phi = 0$ eventually.

Proof of Claim III: By contradiction. Assume Φ is never zero. Then, since Φ is always positive (by Observation V) and does not increase (by Observations II-IV), it converges to a positive number. By Remark 2.1, without loss of generality, Φ is constant and positive. No configuration of Γ is legitimate, hence no configuration is final, by Lemma 8.10. Thus, Γ is infinite. By Lemma 8.9 every process executes Block[2:9] infinitely often. By Claim II, some process is enabled to execute Line 8 and thus eventually executes Block[2:9], hence it executes Line 8, decreasing Φ by Observations II-V, contradiction.

By Claim III, $\mathbb{F}$ eventually holds. □

Proof of Theorem 8.6: Let Γ be a computation of GMD.

By Lemma 8.11, $\mathbb{R}$ eventually holds. By Lemma 8.12, $\mathbb{S}$ eventually holds. By Lemma 8.15, $\mathbb{V}$ eventually holds. By Lemma 8.16, $\mathbb{LV}$ eventually holds. By Lemma 8.17, $\mathbb{F}$ eventually holds, *i.e.,* the configuration is final and legitimate. □

8.3 Time Complexity of GMD

For the remainder of this section, we assume that $\Gamma = \gamma_0 \dots$ is a computation of GMD. When we give a line number, we refer to Algorithm 7.5. Recall also that $K = (2k+3)\left\lceil \frac{N+1}{2k+3} \right\rceil = O(N+k)$.

From Theorem 8.6, we know that GMD is silent and self-stabilizing. We now prove the round complexity of GMD.

Theorem 8.18 *The round complexity of* GMD *is* $O(N+k)$.

We give the proof of Theorem 8.18 below.

Lemma 8.19 *The predicate* $\mathbb{R}$ *holds within* $O(N)$ *rounds.*

Proof: If the configuration does not satisfy $\mathbb{R}$, then *Unison_Error*$(x) =$ TRUE for some x, which implies that x is enabled to execute Line 3, and will do so within one round. By [26, 27], the number of times a reset action can be executed is $O(N)$. Thus, $\mathbb{R}$ holds within $O(N)$ rounds. □

Lemma 8.20 *If* $\mathbb{R}$ *holds, then the predicate* \$ *holds within* $O(N)$ *rounds.*

Proof: Suppose $\mathbb{R}$ holds and \$ does not hold. Let $Min\_r = \min\{x.r\}$. Then $-\alpha \leq Min\_r < 0$. let $M = \{x : x.r = Min\_r\}$. Every member of M is enabled to execute Line 5, and will do so within one round. Thus, *Min_r* increases during the next round. Within $\alpha = N - 2 = O(N)$ rounds, $Min\_r = 0$, which means that \$ holds. □

Lemma 8.21 *Suppose* $(C-1)K < \varrho^* \leq CK$ *for integers* C *and* ϱ^**, and suppose* x^* *is a process and* γ *is a safe configuration such that* $\varrho(x^*) = \varrho^*$ *at* γ*. Then within* $4K$ *rounds,* $\varrho(x) \geq CK$ *for every process* x*.*

Proof: Define $\Phi = \max\{\phi(x)\}$, where $\phi(x) = \begin{cases} 0 \text{ if } \varrho(x) \geq CK \\ 2(CK - \varrho(x)) + K - ||x, x^*|| \text{ otherwise} \end{cases}$

Claim I: If $\varrho(x) < CK$, then $0 < \phi(x) \leq 4K$.

Proof of Claim I: Let x be any process where $\varrho(x) < CK$. Then

$$\phi(x) \geq 2(CK - \varrho(x)) + K - K > 0 \text{ and}$$
$$\phi(x) \leq 2(CK - (\varrho^* - ||x, x^*||)) + K - ||x, x^*|| = 2(CK - \varrho^*) + K + ||x, x^*|| \leq 2K + K + K$$

Claim II: If $\phi(x) = \Phi > 0$, then *Can_Tick*$(x) =$ TRUE.

Proof of Claim II: By contradiction. Suppose that $\varrho(y) < \varrho(x)$ for some $y \in N(x)$. Then $\varrho(y) = \varrho(x) - 1$. By the triangle inequality, we have

$$\phi(y) - \phi(x) = 2(\varrho(x) - \varrho(y)) + ||x, x^*|| - ||y, x^*|| \leq 2(\varrho(x) - \varrho(y)) - ||x, y|| = 1$$

hence $\phi(y) > \Phi$, contradiction.

Claim III: If $\phi(x) = \Phi > 0$, then *Must*$\mathcal{U}(x) =$ TRUE.

Proof of Claim I: Consider the two following cases.

Case 1: $x = x^*$. If $x^*.r = 0$, then $\phi(x^*) = 0$, contradiction. Thus $x^*.r \neq 0$, hence *Must_Tick*(x^*) by Claim II.

Case 2: $x \neq x^*$. Pick $y \in N(x)$ on the path from x to x^*. We need to show that $\varrho(y) > \varrho(x)$. Suppose $\varrho(y) = \varrho(x)$. Then $\phi(y) - \phi(x) = ||x, x^*|| - ||y, x^*|| = 1$, contradiction. Thus, *Must_Tick*(x), which implies *Must*$\mathcal{U}(x)$.

If $\Phi > 0$, then, by Claim III, every x for which $\phi(x) = \Phi$ executes Line 8, decreasing $\phi(x)$, within one round. Thus Φ decreases within one round. Within $4K$ rounds, $\Phi = 0$, and we are done. □

Lemma 8.22 *If* $x.r = 0$ *for all* x *and* $\mathbb{V}$ *does not hold, then some process will execute Line 8 within* $O(k)$ *rounds.*

Proof: By Remark 6.1 and Theorem 5.4, either some process will execute Line 8 or the proof labeling scheme $\mathcal{L}$ will converge within $O(k)$ rounds. If all values of $x.r$ are still zero, there will be a process x where $\mathcal{L}$_Decoder$(x) =$ FALSE. Within one more round, $x.f = 0$, which implies that *Finished*$(x) =$ FALSE. Within one more round, x will execute Line 8. □

Lemma 8.23 *If* \$ *holds, then the predicate* $\mathbb{V}$ *holds within* $O(N + k)$ *rounds.*

Proof: Let $C = \left\lceil \frac{T(2k+3)}{K} \right\rceil$, the smallest integer such that $CK \geq T(2k + 3)$. By Lemma 6.11, $C = O(1)$.

By Lemma 8.21, $\varrho(x) \geq 0$ within $4K$ rounds after \$ first holds. By Lemmas 8.22 and 8.21, it takes $O(K)$ rounds for the minimum value of ϱ to increase from one multiple of K to the next. Thus, $\varrho(x) \geq CK$ for all x within $O(CK) = O(K) = O(N + k)$ rounds. When that bound is achieved, $\mathbb{V}$ holds. □

Lemma 8.24 *If* $\mathbb{V}$ *holds, then the predicate* $\mathbb{LV}$ *holds within* $O(k)$ *rounds.*

Proof: By Remark 6.1 and Theorem 5.4. □

Lemma 8.25 *If* $\mathbb{LV}$ *holds, then the configuration is legitimate and final within* $O(N + k)$ *rounds.*

Proof: Suppose $\mathbb{LV}$ holds. Within $O(K)$ rounds, $x.f = 1 + 2K$ for all x; thereafter, Line 12 will not be executed. Let $C' = \left\lceil \frac{\max\{\varrho(x) \,:\, x \in G\}}{K} \right\rceil$. By Lemma 8.21, $\varrho(x)$ will achieve the value $C'K$ in some configuration within the next $4K$ rounds, for all x. Once $\varrho(x) = C'K$, x cannot execute again, since $Must\_\mathcal{U}(x) = \text{FALSE}$ and $Finished(x) = \text{TRUE}$. By Lemma 8.21, $\mathbb{F}$ is achieved within $O(K) = O(N + k)$ additional rounds. □

Proof Theorem 8.18: We simply add the time complexities given by Lemmas 8.19, 8.20, 8.23, 8.24, and 8.25. □

Notice that since identifiers are unique, a leader can be elected in $O(n)$ rounds [29, 30, 31], after which the value of n can be computed in $O(n)$ additional rounds, hence $N = n$ without loss of generality.

9 Conclusion

We have given a generalization of the problem of finding a minimal k-dominating set of a network. Then, we gave a distributed silent self-stabilizing algorithm solving any instance of the problem. Our algorithm design makes no assumption about the topology of the network, but assumes that processes have unique identifiers.

Our algorithm is implemented in the locally shared memory model with composite atomicity assuming the distributed unfair daemon (the weakest scheduling assumption of the model), has a round complexity $O(N + k)$, and has space complexity $O(\log N + \log k)$. Our algorithm is order-invariant in the sense of Naor and Stockmeyer [14, 15].

As a future research topic, it is worth investigating whether there exists a self-stabilizing (silent) algorithm that computes a k-dominating set of an arbitrary (identified) network which is a good approximation of the minimum one. That is, a self-stabilizing (silent) algorithm that computes a k-dominating set of size s such that $s \leq C \cdot s_{opt}$ where C is a constant (the competitiveness) and s_{opt} is the size of the minimum k-dominating set.

References

[1] A. K. Datta, S. Devismes, L. L. Larmore, A Self-stabilizing $O(n)$-Round k-Clustering Algorithm, in: 28th IEEE Symposium on Reliable Distributed Systems (SRDS 2009), Niagara Falls, New York, USA, September 27-30, 2009, IEEE Computer Society, 2009, pp. 147–155.

[2] E. W. Dijkstra, Self-stabilizing Systems in Spite of Distributed Control, Commun. ACM 17 (11) (1974) 643–644.

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

[4] C. Johnen, Fast, Silent Self-stabilizing Distance-k Independent Dominating Set Construction, Inf. Process. Lett. 114 (10) (2014) 551–555.

[5] E. Caron, A. K. Datta, B. Depardon, L. L. Larmore, A Self-stabilizing K-Clustering Algorithm Using an Arbitrary Metric, in: Euro-Par 2009 Parallel Processing, 15th International Euro-Par Conference, Delft, The Netherlands, August 25-28, 2009. Proceedings, Vol. 5704 of Lecture Notes in Computer Science, Springer, 2009, pp. 602–614.

[6] E. Caron, A. K. Datta, B. Depardon, L. L. Larmore, A Self-stabilizing k-Clustering Algorithm for Weighted Graphs, J. Parallel Distrib. Comput. 70 (11) (2010) 1159–1173.

[7] A. K. Datta, L. L. Larmore, P. Vemula, A Self-stabilizing $O(k)$-Time k-Clustering Algorithm, The Computer Journal 53 (2010) 342–350.

[8] A. K. Datta, S. Devismes, K. Heurtefeux, L. L. Larmore, Y. Rivierre, Competitive Self-stabilizing k-Clustering, Theor. Comput. Sci. 626 (2016) 110–133.

[9] Y. Fernandess, D. Malkhi, K-clustering in Wireless Ad Hoc Networks, in: Proceedings of the 2002 Workshop on Principles of Mobile Computing, POMC 2002, October 30-31, 2002, Toulouse, France, ACM, 2002, pp. 31–37.

[10] S. Kutten, D. Peleg, Fast Distributed Construction of Small k-Dominating Sets and Applications, J. Algorithms 28 (1) (1998) 40–66.

[11] M. A. Spohn, J. J. Garcia-Luna-Aceves, Bounded-Distance Multi-Clusterhead Formation in Wireless Ad Hoc Networks, Ad Hoc Networks 5 (2004) 504–530.

[12] F. Kuhn, T. Moscibroda, R. Wattenhofer, Fault-Tolerant Clustering in Ad Hoc and Sensor Networks, in: 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06), 2006, pp. 68–68.

[13] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

[14] A. Korman, S. Kutten, D. Peleg, Proof Labeling Schemes, Distributed Computing 22 (4) (2010) 215–233.

[15] M. Naor, L. J. Stockmeyer, What Can be Computed Locally?, SIAM J. Comput. 24 (6) (1995) 1259–1277.

[16] A. K. Datta, L. L. Larmore, S. Devismes, K. Heurtefeux, Y. Rivierre, Self-stabilizing Small k-Dominating Sets, International Journal of Networking and Computing 3 (1) (2013) 116–136.

[17] M. Ikeda, S. Kamei, H. Kakugawa, A Space-Optimal Self-stabilizing Algorithm for the Maximal Independent Set Problem, in: Third International Conference on Parallel and Distributed Computing, Applications and Technologies, Kanazawa, Japan, 2002, pp. 70–74.

[18] H. Kakugawa, T. Masuzawa, A Self-stabilizing Minimal Dominating Set Algorithm with Safe Convergence, in: 20th International Parallel and Distributed Processing Symposium (IPDPS), IEEE, Rhodes Island, Greece, 2006, pp. 263–270.

[19] S. Shukla, D. J. Rosenkrantz, S. S. Ravi, Observations on Self-stabilizing Graph Algorithms on Anonymous Networks, in: Second Workshop on Self-Stabilizing Systems, Las Vegas, Nevada, 1995, pp. 7.1–7.15.

[20] A. K. Datta, S. Devismes, L. L. Larmore, Technical Note on the Round Complexity of the MIS Algorithm of Kakugawa and Masuzawa, Online: `http://www-verimag.imag.fr/~devismes/notes/KM06c.pdf`, technical note (March 2018).

[21] V. Drabkin, R. Friedman, M. Gradinariu, Self-stabilizing Wireless Connected Overlays, in: A. A. Shvartsman (Ed.), Principles of Distributed Systems, 10th International Conference, OPODIS 2006, Bordeaux, France, December 12-15, 2006, Proceedings, Vol. 4305 of Lecture Notes in Computer Science, Springer, 2006, pp. 425–439.

[22] S. Kamei, H. Kakugawa, A Self-stabilizing Approximation for the Minimum Connected Dominating Set with Safe Convergence, in: T. P. Baker, A. Bui, S. Tixeuil (Eds.), Principles of Distributed Systems, 12th International Conference, OPODIS 2008, Luxor, Egypt, December 15-18, 2008. Proceedings, Vol. 5401, Springer, 2008, pp. 496–511.

[23] C. Johnen, L. H. Nguyen, Robust Self-stabilizing Weight-based Clustering Algorithm, Theor. Comput. Sci. 410 (6-7) (2009) 581–594.

[24] N. Mitton, B. Sericola, S. Tixeuil, E. Fleury, I. G. Lassous, Self-stabilization in Self-organized Wireless Multihop Networks?, Ad Hoc & Sensor Wireless Networks 11 (1-2) (2011) 1–34.

[25] S. Dolev, A. Israeli, S. Moran, Uniform Dynamic Self-stabilizing Leader Election, IEEE Trans. Parallel Distrib. Syst. 8 (4) (1997) 424–440.

[26] C. Boulinier, F. Petit, V. Villain, When Graph Theory Helps Self-stabilization, in: S. Chaudhuri, S. Kutten (Eds.), Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 2004, ACM, 2004, pp. 150–159.

[27] C. Boulinier, F. Petit, V. Villain, Synchronous vs. Asynchronous Unison, Algorithmica 51 (1) (2008) 61–80.

[28] K. Altisen, S. Devismes, On Probabilistic Snap-stabilization, Theor. Comput. Sci. 688 (2017) 49–76.

[29] A. K. Datta, L. L. Larmore, P. Vemula, An O(n)-time Self-stabilizing Leader Election Algorithm, J. Parallel Distrib. Comput. 71 (11) (2011) 1532–1544.

[30] A. K. Datta, L. L. Larmore, P. Vemula, Self-stabilizing Leader Election in Optimal Space under an Arbitrary Scheduler, Theor. Comput. Sci. 412 (40) (2011) 5541–5561.

[31] K. Altisen, A. Cournier, S. Devismes, A. Durand, F. Petit, Self-stabilizing Leader Election in Polynomial Steps, Inf. Comput. 254 (2017) 330–366.

List of Symbols

$x.id$ the unique identifier of process x, page 3

$\gamma \mapsto \gamma'$ A *step* from configuration γ to configuration γ', page 4

$\Gamma = \gamma_0\gamma_1\ldots$ a *computation*, each $\gamma_i \mapsto \gamma_{i+1}$ is a step, page 4

$N(x)$ the *set of neighbors* of process x, page 5

$||x, y||$ the distance from x to y, page 5

$N_d(x)$ the *d-hop neighborhood* of x, page 5

$N_1(x)$ $= N(x) \cup \{x\}$, page 5

$N_d(S)$ $= \bigcup \{N_d(x) : x \in S\}$, page 5

$||x, S||$ $= \min\{||x, s|| : s \in S\}$, page 5

$||x, \emptyset||$ $= \infty$, page 5

$||x, S||_2$ the distance from x to its *second* closest member of S, page 5

$\mathit{Excl\_Followers}_{k,D}(x)$ the set of all k-exclusive followers of x relative to D, page 5

D is *k-exclusive* D is *k-exclusive* if $\mathit{Excl\_Followers}(x)$ is non-empty for all $x \in D$, page 5

GMDS *generalized minimal k-dominating set* problem, page 6

(G, k, R^*, D^*) instance of GMDS: G is a graph, k is an integer, and $R^* \subseteq D^* \subseteq G$, page 6

$\mathit{Closest}_D(x)$ nodes which are strictly closer to x than to any other node of D, page 6

augmented ID type, either $\perp$, or the ordered pair $x.\widetilde{id} = (\mathit{rank}(x), x.id)$, page 17

$x.\widetilde{id}$ the *augmented identifier* of process x, page 17

$\min S$ the process in S of minimum augmented identifier, page 17

$\max S$ the process in S of maximum augmented identifier, page 17

$\max \emptyset$ $= \perp$, page 17

$\mathit{Leader}_{D,R}(x)$ $= \min\{y \in N_k(x)\}$, page 17

$\mathit{Followers}_{D,R}(x)$ $= \{y : \mathit{Leader}_{D,R}(y) = x\}$, page 17

$y \ominus x$ the *(path) delay* from x to y, page 28

$\mathbb{R}$ true if $\mathit{Near}(x.r, y.r)$ for all x and y such that $x.r > 0$ and $y \in N(x)$, page 30

$\mathbb{S}$ true if the configuration is *safe*, *i.e.*, $\mathcal{U}\_\mathit{Safe}(x)$ holds for all x, page 30

$\varrho : G \to \mathbb{Z}$ a *lifting* of clock r, an integral function such that $\varrho(x) - \varrho(y) = x \ominus y$ for all x, y, and such that $\varrho(x)$ increments each time $x.r$ increments, page 30

$V(x)\langle i\rangle$ the value of $V(x)$ at the first configuration at which $\varrho(x) = i$, page 31

$\mathbb{V}$ true if $\mathbb{S}$ holds and $\mathcal{D}$ has reached a legitimate configuration, page 32

$\mathbb{LV}$ true if $\mathbb{V}$ holds, and for all x, $\mathcal{L}$_Decoder$(x) = \textsc{true}$ and $F(x) = 2K + 1$, page 32

$\mathbb{F}$ true if the configuration is final, page 32