Competitive Self-Stabilizing k-Clustering

Ajoy K. Datta, Stéphane Devismes, Karel Heurtefeux, Lawrence L. Larmore, Yvan Rivierre

Verimag Research Report n^o TR-2011-16

November 17, 2011

Competitive Self-Stabilizing k-Clustering

*Ajoy K. Datta, Stéphane Devismes, Karel Heurtefeux, Lawrence L. Larmore,
Yvan Rivierre*

November 17, 2011

Abstract

A k-*cluster* of a graph is a connected non-empty subgraph C of radius at most k, *i.e.*, all members of C are within distance k of a particular node of C, called the *clusterhead* of C. A k-*clustering* of a graph is a partitioning of the graph into distinct k-clusters. Finding a minimum cardinality k-*clustering* is known to be $\mathcal{NP}$-hard.
In this paper, we propose a silent self-stabilizing asynchronous distributed algorithm for constructing a k-clustering of any connected network with unique IDs. Our algorithm stabilizes in $O(n)$ rounds, using $O(\log n)$ space per process, where n is the number of processes. In the general case, our algorithm constructs $O(\frac{n}{k})$ k-clusters. If the network is a Unit Disk Graph (UDG), then our algorithm is $7.2552k + O(1)$-*competitive*, that is, the number of k-clusters constructed by the algorithm is at most $7.2552k + O(1)$ times the minimum possible number of k-clusters in any k-clustering of the same network. More generally, if the network is an Approximate Disk Graph (ADG) with approximation ratio λ, then our algorithm is $7.2552\lambda^2 k + O(\lambda)$-*competitive*.
Our solution is based on the self-stabilizing construction of a data structure called the *MIS Tree*, a *spanning tree* of the network whose processes at even levels form a maximal independent set of the network. The MIS tree construction is the time bottleneck of our k-clustering algorithm, as it takes $\Theta(n)$ rounds in the worst case, while the remainder of the algorithm takes $O(\mathcal{D})$ rounds, where $\mathcal{D}$ is the diameter of the network. We would like to improve that time to be $O(\mathcal{D})$, but we show that our distributed MIS tree construction is a $\mathcal{P}$-*complete* problem.

Keywords: self-stabilization, maximal independent set, MIS tree, k-clustering, competitiveness

1 Introduction

Consider a simple connected undirected graph $G = (V, E)$, where V is a set of n nodes and E a set of edges. For any nodes p and q, we define $\|p, q\|$, the *distance* from p to q, to be the length of the shortest path in G from p to q. Given a non-negative integer k, a *k-cluster* of G is defined to be a set $C \subseteq V$, together with a designated node $Clusterhead(C) \in C$, such that each member of C is within distance k of $Clusterhead(C)$. A *k-clustering* of G is a partition of V into distinct k-clusters.

A major application of *k-clustering* is in the implementation of an efficient routing scheme in a network of processes. Indeed, we could use the rule that a process that is not a clusterhead, communicates only with processes in its own k-cluster, and that clusterheads communicate with each other *via* virtual “super-edges,” implemented as paths in the network.

Ideally, we would like to find a k-clustering with the minimum number of k-clusters. However, this problem is known to be $\mathcal{NP}$-hard [15]. Instead, we propose here an asynchronous distributed silent self-stabilizing algorithm to construct $O(\frac{n}{k})$ k-clusters in any arbitrary network with unique IDs. If the network is a Unit Disk Graph (UDG), then our algorithm is $7.2552k + O(1)$-*competitive*, that is, it builds a k-clustering which has at most $7.2552k + O(1)$ times as many clusters as the minimum cardinality k-clustering.

Related Work *Self-stabilization* [9] is a versatile property, enabling an algorithm to withstand transient faults in a distributed system. A self-stabilizing algorithm, after transient faults hit and place the system in some arbitrary state, enables the system to recover without external (*e.g.*, human) intervention in finite time.

There are several known asynchronous self-stabilizing distributed algorithms for finding a k-clustering of a network, *e.g.*, [7, 6, 3]. The solution in [7] stabilizes in $O(k)$ rounds using $O(k \log n)$ space per process. The algorithm given in [6] stabilizes in $O(n)$ rounds using $O(\log n)$ space per process. The algorithm given in [3] stabilizes in $O(kn)$ rounds using $O(k \log n)$ space per process.

In [**?**], an asynchronous silent self-stabilizing algorithm is given which computes a *k-dominating* set of at most $\lfloor \frac{n}{k+1} \rfloor$ processes. A set of vertices D of G is called *k-dominating* if every vertex of G is within k hops of some member of D. Hence, the set of clusterheads of a k-clustering is a k-dominating set. Then, any k-dominating set can be used to construct a k-clustering by letting each member of the set be a clusterhead, and letting each process join the nearest clusterhead. The k-dominating set construction given in [**?**] stabilizes in $O(n)$ rounds using $O(\log n + k \log \frac{n}{k})$ bits per process.

Note that all these aforementioned algorithms (*i.e.*, [7, 6, 3, **?**]) are written in the shared memory model and none of them is *competitive*.

There are several *non self-stabilizing* distributed solutions for finding a k-clustering of a network [1, 13, 19, 20]. Of those, only [13] deals with competitiveness. Moreover, they are all written in message-passing model. Deterministic solutions given in [1, 13] are designed for *asynchronous mobile ad hoc* networks, *i.e.*, they assume networks with a UDG topology. The time and space complexities of the solution in [1] are $O(k)$ and $O(k \log n)$, respectively. Spohn and Garcia-Luna-Aceves [20] give a distributed solution to a more 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. The time and space complexities of this algorithm for asynchronous networks are not given. Ravelomanana [19] gives a randomized algorithm for synchronous UDG networks whose time complexity is $O(\mathcal{D})$ rounds, where $\mathcal{D}$ is the diameter of the network. Fernandess and Malkhi [13] give a k-clustering algorithm that takes $O(n)$ steps using $O(\log n)$ memory per process, provided a BFS tree of the network is already given. In the special case that the network is a UDG, their algorithm is $8k + O(1)$-*competitive*. [1]

Contributions In this paper, we give a silent self-stabilizing asynchronous distributed algorithm for constructing a k-clustering in any connected network with unique IDs. Our algorithm stabilizes in $O(n)$ rounds using $O(\log n)$ space per process. In the general case, our algorithm constructs at most $1 + \lfloor \frac{n-1}{k+1} \rfloor$ k-clusters. If the network is a UDG, then our algorithm is $7.2552k + O(1)$-*competitive*, that is, the number

[1]Actually, in [13], a k-cluster is defined to have diameter at most k, while the definition in this paper uses radius k. They give competitiveness $4k + O(1)$, which is equivalent to competitiveness $8k + O(1)$ using our definition of k-cluster.

of k-clusters constructed by the algorithm is at most $7.2552k + O(1)$ times the minimum possible number of k-clusters in any k-clustering of the same network. This result is an improvement over that of [13]. More generally, if the network is an Approximate Disk Graph (ADG) with approximation ratio λ, then our algorithm is $7.2552\lambda^2 k + O(\lambda)$-competitive. UDG and ADG are commonly used to model the topology of wireless ad hoc networks.

Our solution is based on the self-stabilizing construction of a data structure called an *MIS Tree*, a spanning tree of the network whose processes at even levels form a maximal independent set of the network. The MIS tree method was introduced by Fernandess and Malkhi [13]. The MIS tree construction is the time bottleneck of our k-clustering algorithm, as it takes $\Theta(n)$ rounds in the worst case, and the remainder of the algorithm takes $O(\mathcal{D})$ rounds, where $\mathcal{D}$ is the diameter of the network. We would like to improve that time to be $O(\mathcal{D})$, however, that will most likely involve different techniques, since whether a given process is part of the Fernandess-Malkhi MIS is a $\mathcal{P}$*-complete* problem, as we show in Section 6.

Roadmap In the next section, we present the model used throughout this paper. In Section 3, we give our self-stabilizing MIS tree construction. In Section 4, we give our self-stabilizing k-clustering algorithm. In Section 5, we analyze the competitiveness of our k-clustering algorithm in UDGs and ADGs. In Section 6, we show that the problem we solved in Section 3 is $\mathcal{P}$*-complete*. Finally, in Section 7, we give some perspectives.

2 Preliminaries

Computational Model Consider a simple connected bidirectional network $G = (V, E)$ where V is a set of n processes and E a set of links. Processes have unique IDs. By an abuse of notation, we shall identify any process with its ID, whenever convenient.

We assume the *shared memory model* of computation [9], where a process p can read its own variables and those of its neighbors, but can write only to its own variables. Let $\mathcal{N}_p$ denote the set of neighbors of p. Each process operates according to its (local) *program*. We call *(distributed) algorithm* $\mathcal{A}$ a collection of n *programs*, each one operating on a single process. The *program* of each process is a finite set of actions: $\langle label \rangle \ :: \ \langle guard \rangle \ \longrightarrow \ \langle statement \rangle$. *Labels* are only used to identify actions. The *guard* of an action in the program of a process p is a Boolean expression involving the variables of p and its neighbors. The *statement* of an action of p updates one or more variables of p. 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 in $\mathcal{A}$. A *configuration* of $\mathcal{A}$ is an instance of the states of processes in $\mathcal{A}$. We denote by $\gamma(p)$ the state of process p in configuration γ.

Let $\mapsto$ be the binary relation over configurations of $\mathcal{A}$ such that $\gamma \mapsto \gamma'$ if and only if it is possible for the network to change from configuration γ to configuration γ' in one step of $\mathcal{A}$. An *execution* of $\mathcal{A}$ is a maximal sequence of its configurations $e = \gamma_0\gamma_1 \dots \gamma_i \dots$ such that $\gamma_{i-1} \mapsto \gamma_i$ for all $i > 0$. The term "maximal" means that the execution is either infinite, or ends at a *terminal* configuration in which no action of $\mathcal{A}$ is enabled at any process. Each step $\gamma_i \mapsto \gamma_{i+1}$ consists of one or more enabled processes executing an action. The evaluations of all guards and executions of all statements of those actions are presumed to take place in one atomic step; this model is called *composite atomicity* [10].

We assume that each step from a configuration to another is driven by a *scheduler*, also called a *daemon*. If one or more processes are enabled, the scheduler selects at least one of these enabled processes to execute an action. A scheduler may have some *fairness* properties. Here, we assume a *weakly fair* scheduler, *i.e.*, it allows every *continuously* enabled process to eventually execute an action.

We say that a process p is *neutralized* in the step $\gamma_i \mapsto \gamma_{i+1}$ if p is enabled in γ_i and not enabled in γ_{i+1}, but does not execute any action between these two configurations. The neutralization of a process represents the following situation: at least one neighbor of p changes its state between γ_i and γ_{i+1}, and this change effectively makes the guard of all actions of p false.

To evaluate the time complexity, we use the notion of *round* [12]. The first *round* of an execution ϱ, noted ϱ', is the minimal prefix of ϱ in which every process that is enabled in the initial configuration either executes an action or becomes neutralized. Let ϱ'' be the suffix of ϱ starting from the last configuration of

ϱ'. The second round of ϱ is the first round of ϱ'', the third round of ϱ is the second round of ϱ'', and so forth.

Self-Stabilization and Silence A configuration *conforms* to a predicate if the predicate is satisfied in the configuration; otherwise the configuration *violates* the predicate. By this definition every configuration conforms to predicate *true* and none conforms to predicate *false*. Let R and S be predicates on configurations of the algorithm. Predicate R is *closed* with respect to the algorithm actions if every configuration of any execution of the algorithm that starts at a configuration conforming to R also conforms to R. Predicate R *converges* to S if R and S are closed and every execution starting from a configuration conforming to R contains a configuration conforming to S. A distributed algorithm is *self-stabilizing with respect to* predicate R if *true* converges to R. An algorithm is *silent* [11] if each of its executions is finite. In other words, starting from an arbitrary configuration, the network will eventually reach a configuration where none of its actions is enabled at any process.

Composition To simplify the design of our algorithm, we use *hierarchical collateral composition* [**?**] which is a variant of *collateral composition* [21]. When we collaterally compose two algorithms $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A}$ and $\mathcal{B}$ run concurrently and $\mathcal{B}$ uses the outputs of $\mathcal{A}$ in its computations. In the variant we use, we modify the code of $\mathcal{B}$ so that a process executes an action of $\mathcal{B}$ only when it has no enabled action in $\mathcal{A}$.

Definition 1 *Let $\mathcal{A}$ and $\mathcal{B}$ be two algorithms such that no variable written by $\mathcal{B}$ appears in $\mathcal{A}$. The* hierarchical collateral composition *of $\mathcal{A}$ and $\mathcal{B}$, noted $\mathcal{B} \circ \mathcal{A}$, is the algorithm defined as follows: (i) $\mathcal{B} \circ \mathcal{A}$ contains all variables of $\mathcal{A}$ and $\mathcal{B}$; (ii) $\mathcal{B} \circ \mathcal{A}$ contains all actions of $\mathcal{A}$; (iii) For every action $G_i \to S_i$ of $\mathcal{B}$, $\mathcal{B} \circ \mathcal{A}$ contains the action $\neg C \wedge G_i \to S_i$ where C is the disjunction of all guards of actions in $\mathcal{A}$.*

We recall a theorem from [**?**] that gives sufficient conditions to show the correctness of an algorithm obtained by hierarchical collateral composition.

Theorem 1 *$\mathcal{B} \circ \mathcal{A}$ is self-stabilizing* w.r.t *predicate* SP *under a weakly fair scheduler if: (i) $\mathcal{A}$ is silent algorithm under a weakly fair scheduler, and (ii) $\mathcal{B}$ converges to* SP *from any terminal configuration of $\mathcal{A}$ under a weakly fair scheduler.*

3 The MIS Tree

In this section, we first recall the data structure *MIS tree* (for Maximal Independent Set tree), introduced in [13]. We define an MIS tree to be a spanning tree rooted at a given node r, where the set of all nodes at even levels is a maximal independent set of the network. This data structure has interesting properties that will be used to compute a competitive k-clustering, when the network is a UDG. In the second part of the section, we give a self-stabilizing algorithm that computes an MIS tree in any arbitrary identified network within $O(n)$ rounds. There could be many different MIS trees for a given network and a given r; the one we construct has the same specification as that constructed in [13]. We leave open the possibility that there could be faster algorithm to compute an MIS tree, but, in Section 6, we will prove that, if there is a distributed algorithm which constructs the specific MIS tree constructed here in $O(\mathcal{D})$ time, then $\mathcal{P} = \mathcal{NC}$ (Nick's Class), which would be as startling as $\mathcal{P} = \mathcal{NP}$.

3.1 Definition of MIS Tree

Suppose $G = (V, E)$ is a connected undirected graph. A set $I \subseteq V$ is an *independent set* of G if no two distinct members of I are neighbors in G. An independent set I of G is *maximal* if no proper superset of I is an independent set of G. A *spanning tree* of G is any connected graph $T = (V_T, E_T)$ such that $V_T = V$, $E_T \subseteq E$ and $|E_T| = |V_T| - 1$. Any spanning tree becomes a rooted tree by choosing a distinguished root r; in this paper, all spanning trees are rooted.

Given a rooted spanning tree T, the *level* of node p, $\texttt{Level}(p)$, is defined to be its distance to the root r. The *height* of T, noted $h(T)$, is $\max_{p \in V_T} \texttt{Level}(p)$. Let $T(p)$ be the subtree of T rooted at any given node p, and define $h(T(p))$ to be the height $T(p)$. The *parent* of p in T is p itself if $p = r$, otherwise it is its unique neighbor q in T such that $h(p) = h(q) + 1$.

Definition 2 *An* MIS tree *T of G is a spanning tree of G rooted at some node r such that the set of nodes at even levels of T is a maximal independent set of G.*

Property 1 *Let T be an MIS tree of G. Let I be the maximal independent set formed by the nodes at even levels of T. If σ is a path of T of length ℓ (*i.e.*, $\ell + 1$ nodes), then σ contains at least $\lceil \frac{\ell}{2} \rceil$ members of I.*

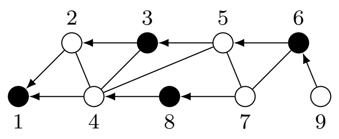


Figure 1: Example of LFMIST.

Assume that an ordering $p_1, p_2, \ldots, p_n$ of V is given. Any rooted tree T of G can be encoded as an n-tuple of numbers in the range $1..n$, as follows. The i^{th} entry of the encoding of T is j if p_j is the parent of p_i in T. The *lexically first MIS tree* (LFMIST) of G with root r is then defined to be that MIS tree of G whose is first in the lexical order of the encodings of all MIS trees of G with root r. For example, in Figure 1, the members of the maximal independent set are shown in black and the encoding of the tree is $(1, 1, 2, 1, 3, 5, 8, 4, 6)$.

3.2 The Algorithm to construct an MIS Tree

Our self-stabilizing algorithm to construct an MIS tree is a hierarchical collateral composition of two algorithms: $\mathcal{MIST} \circ \mathcal{BFST}$. Algorithm $\mathcal{BFST}$ constructs a breadth-first spanning tree (BFS tree). Then, $\mathcal{MIST}$ uses the BFS tree to compute an MIS Tree of the network in $O(n)$ rounds.

Algorithm $\mathcal{BFST}$ We define a *breadth first spanning tree* (BFS tree) rooted at r, for a graph $G = (V, E)$ to be any spanning tree T rooted at r such that the path, through T, from any node p to r has length $\|p, r\|$ (the distance from p to r in G).

Let $\mathcal{BFST}$ be any silent self-stabilizing breadth-first spanning tree algorithm for a network with unique IDs which works under a weakly fair scheduler. That is, starting from an arbitrary configuration, $\mathcal{BFST}$ converges to a terminal configuration where a root r and a breadth-first spanning tree of the G, rooted at r, is output. Henceforth, we denote by $\texttt{Level}_{\texttt{BFS}}(p)$ the level of any process p in the breadth-first spanning tree computed by $\mathcal{BFST}$.

Many silent self-stabilizing breadth-first search spanning tree algorithms have been given in the literature. See [16] for one of the first papers on that topic. This algorithm was designed for arbitrary rooted networks, but it can be easily adapted to work in arbitrary network with unique IDs by composing it with a leader election algorithm, *e.g.*, [8]. The composition of these two latter algorithms stabilizes in $O(n)$ rounds uses $O(\log n)$ space per process.

Algorithm $\mathcal{MIST}$ Let r be the root of the BFS tree computed by $\mathcal{BFST}$. Let $\prec$ be an order on processes defined as follows : $p \prec q$ if and only if $(\|p, r\|, p)$ is smaller than $(\|q, r\|, q)$ in the lexical ordering of the pairs. Using the outputs of $\mathcal{BFST}$, $\mathcal{MIST}$ computes an MIS tree of the network that is lexically first *w.r.t.* to $\prec$. The formal description of $\mathcal{MIST}$ is given in Algorithm 1. In $\mathcal{MIST}$, the program of each process p contains two variables:

- The Boolean variable *p.dominator*, which determines if p is in the independent set or not.
- The pointer variable *p.parent*, which points to the parent of p in the MIS tree.

Every process p such that *p.dominator* $=$ *true* is said to be a *dominator*, otherwise it is said to be *dominated*. Eventually, the set $\{p \in V \mid p.dominator\}$ is fixed and forms a maximal independent set of the network thanks to Action SetDominator.

To decide of its status dominator/dominated, each process uses a *priority*, noted $Priority(p)$, which is defined by the tuple $(\texttt{Level}_{\texttt{BFS}}(p), p)$ (*n.b.*, $\texttt{Level}_{\texttt{BFS}}(p)$ is eventually equal to the distance of p to the root of the BFS tree). According to the priorities and the status of its neighbors, p decides its status as follows: p is a dominator if and only if all its neighbors q are either dominated or satisfy $Priority(q) > Priority(p)$, where $>$ is the strict lexical ordering. According to this rule, the root of the BFS tree is the node of minimum priority and consequently is eventually definitely a dominator. All its neighbors becomes dominated, and so on.

Each process must choose a parent such that the parent links form a spanning tree, and the set of processes at even levels is exactly the set of dominators. The root r sets its parent variable to r. All other processes choose as parent the neighbor having a status different of their own of minimum priority. This forces a strict alternation between status dominator/dominating along every path of the tree. As the root is at level zero and of dominating status, this alternation makes the tree an MIS tree.

Algorithm 1 $\mathcal{MIST}$, code for each process p

```
Inputs: $\texttt{Level}_{\texttt{BFS}}(p) \in \mathbb{N}$
Variables: $p.dominator$: Boolean ; $p.parent \in \mathcal{N}_p \cup \{p\}$
Macros:
  $Priority(p)$    =  $(\texttt{Level}_{\texttt{BFS}}(p), p)$
  $Dominator(p)$   =  $\forall q \in \mathcal{N}_p, Priority(p) < Priority(q) \vee \neg q.dominator$
  $Parent(p)$      =  if $\texttt{Level}_{\texttt{BFS}}(p) = 0$ then $p$
                         else $q \in \mathcal{N}_p \mid Priority(q) = \min\{Priority(q') \mid q' \in \mathcal{N}_p \wedge q'.dominator \neq p.dominator\}$
Actions:
  SetDominator  ::  $p.dominator \neq Dominator(p)$                          $\longrightarrow$  $p.dominator \leftarrow Dominator(p)$
  SetParent     ::  $p.dominator = Dominator(p) \wedge p.parent \neq Parent(p)$  $\longrightarrow$  $p.parent \leftarrow Parent(p)$
```

Correctness and Complexity Analysis According to Theorem 1, to show the correctness of $\mathcal{MIST} \circ \mathcal{BFST}$, we show that $\mathcal{MIST}$ constructs a MIS tree starting from any configuration where no action of $\mathcal{BFST}$ is enabled. In such a configuration, a BFS tree T_{BFS} rooted at some node is available. In the following, we denote by r the root of T_{BFS}, which will be also the root of the MIS tree.

The following two lemmas show that $\mathcal{MIST}$ stabilizes in $O(n)$ rounds after $\mathcal{BFST}$ has stabilized.

Lemma 1 *Starting from any configuration where no action of $\mathcal{BFST}$ is enabled, all actions SetDominator are disabled forever after at most n rounds.*

Proof. Let γ be a configuration where no action of $\mathcal{BFST}$ is enabled. From γ, $Priority(p)$ is fixed forever for every process p. Let p_1,…,p_n the list of processes ordered by $\prec$ (the lexical ordering *w.r.t.* priorities) in γ. We show the lemma by induction on the rank of every process in the ordering.

- **Base case:** In γ, if p_1.*dominator* $\neq$ *true*, p_1 is continuously enabled to set p_1.*dominator* $=$ *true*. Once, p_1.*dominator* $=$ *true*, action SetDominator is disabled at p_1 forever. So, after at most one round from γ, action SetDominator of p_1 is disabled forever.
- **Inductive Hypothesis:** Let j a positive integer. Assume that for every process p_i such that $i \leq j$, action SetDominator is disabled forever at p_i after at most i rounds from γ.
- **Inductive step:** Consider process p_{j+1} in the first configuration of the $(j+1)^{st}$ round from γ. Every neighbor q of p_{j+1} has priority that is fixed forever; moreover if $Priority(q) < Priority(p_{j+1})$, then the value q.*dominator* is fixed forever by induction hypothesis. So, either action SetDominator is disabled at p_{j+1} or it is continuously enabled. Hence, at the end of the current round, the value of p_{j+1} is fixed forever and the induction holds.

The maximum rank being n, the lemma is verified. □

Lemma 2 *Starting from any configuration where no action of $\mathcal{BFST}$ is enabled, if at least $n+1$ additional rounds have elapsed, no action of $\mathcal{MIST}$ is enabled.*

Proof. Let γ be a configuration where no action of $\mathcal{BFST}$ is enabled. By Lemma 1, after at most n rounds from γ, no action SetDominator is enabled. So, from that point, the values of $Priority(p)$ and $p.dominator$ are fixed forever. Now, for all processes, the guard of action SetParent only depends on these values. So, after at most one additional rounds, no action of $\mathcal{MIST}$ is can ever again be enabled, and we are done. □

We now consider any terminal configuration γ of $\mathcal{MIST} \circ \mathcal{BFST}$. Let I the set of all dominator processes in γ, that is, the set of all processes p such that $p.dominator = true$ in γ.

The following three technical lemmas are used in order to prove Lemma 6 which states the correctness of $\mathcal{MIST} \circ \mathcal{BFST}$.

Lemma 3 *In any terminal configuration γ of $\mathcal{MIST} \circ \mathcal{BFST}$, I is a maximal independent set of the network.*

Proof. Suppose the set I is not independent, then there exist two neighbors p and q such that $p.dominator$ and $q.dominator$. Then, either $Priority(p) < Priority(q)$ or $Priority(q) < Priority(p)$. In the first case, Action SetDominator is enabled at q, in the latter Action SetDominator is enabled at p, contradiction.

Suppose the independent set I is not maximal, then there exists a process p such that $\neg p.dominator$ and for every neighbor q of p, $\neg q.dominator$. Then Action SetDominator is enabled at p, contradiction. □

In γ, r is the only process such that $\texttt{Level}_{\texttt{BFS}}(r) = 0$. By the definition of $Parent(p)$, we then have:

Remark 1 *In γ, for every process p, either $p = r$ and p.parent $= r$ or $p \neq r$ and p.parent $\in \mathcal{N}_p$.*

Lemma 4 *In any terminal configuration γ of $\mathcal{MIST} \circ \mathcal{BFST}$, $Priority(p\text{.parent}) < Priority(p)$. for every process $p \neq r$,*

Proof. We consider two cases, according to the status of p:

- $p \in I$. Then, by Lemma 3, $\forall q \in \mathcal{N}_p$, $q.dominator = false$, in particular for $q = \texttt{Parent}_{\texttt{BFS}}(p)$. Note that $\texttt{Level}_{\texttt{BFS}}(\texttt{Parent}_{\texttt{BFS}}(p)) = \texttt{Level}_{\texttt{BFS}}(p) - 1$. Thus, by definition of the Macro $Parent(p)$, $\texttt{Level}_{\texttt{BFS}}(p.parent) = \texttt{Level}_{\texttt{BFS}}(\texttt{Parent}_{\texttt{BFS}}(p))$. Consequently, $Priority(p.parent).Priority(p)$.
- $p \notin I$. Then $\neg Dominator(p)$. Now, as no two processes have equal priority, we have $\exists q \in \mathcal{N}_p, Priority(p) > Priority(q) \wedge q.dominator$. So, $Priority(p.parent) \leq Priority(q)$ by definition of Macro $Parent(p)$. Consequently, $Priority(p.parent) < Priority(p)$.

□

In the following, we denote by T_{MIS} the subgraph induced by the values of the parent pointers of $\mathcal{MIST}$ in the terminal configuration γ. Formally, $T_{MIS} = (V, E_{MIS})$, where E_{MIS} is the set $\{\{p, p.parent\} \mid p \in V \setminus \{r\}\}$ defined in γ. (Recall that r is the unique process such that $r.parent = r$ in γ, by Remark 1.)

Lemma 5 *In any configuration where no action of $\mathcal{MIST} \circ \mathcal{BFST}$ is enabled, T_{MIS} is a spanning tree of the network.*

Proof. We show by contradiction that T_{MIS} is connected and acyclic:

- Suppose T_{MIS} is not acyclic. Then, there exists a elementary cycle in $C = (c_0, c_1, \ldots, c_m = c_0)$ such that $\forall i \in [0..m-1]$, $c_i.parent = c_{i+1}$ and $m > 0$. By Remark 1, $r \notin C$. By Lemma 4, $\forall i \in [0..m-1]$, $Priority(c_i) < Priority(c_{i+1})$. By transitivity, $Priority(c_0) < Priority(c_m)$, that is $Priority(c_0) < Priority(c_0)$, contradiction.
- Suppose T_{MIS} is not connected, then there exist at least two connected components in T_{MIS}. At least one component, noted G', does not contain the root r. Every process $p \in G'$ has a parent in G', by Macro $Parent(p)$. Hence, there are as many edges as processes in G', *i.e.*, there is a cycle in G'. As T_{MIS} is acyclic, we obtain a contradiction.

$\square$

In the following, we denote by $\texttt{Level}_{\texttt{MIS}}(p)$ the level of any process p in the MIS tree T_{MIS} computed by algorithm $\mathcal{MIST}$.

Lemma 6 *In any configuration where no action of $\mathcal{MIST} \circ \mathcal{BFST}$ is enabled, T_{MIS} is an MIS tree of the network.*

Proof. By Lemma 5, T_{MIS} is a spanning tree of the network. By Lemma 3, I is an MIS of the network. We now show that the even levels of T_{MIS} form I. Formally, we prove that $\texttt{Level}_{\texttt{MIS}}(p)$ is even if and only if $p.dominator$ for all $p \in V$, by induction on $\texttt{Level}_{\texttt{MIS}}(p)$.

First, the root process r is necessarily in I. For the inductive step, let p be a process other than r, and let $L = \texttt{Level}_{\texttt{MIS}}(p) > 0$. By the inductive hypothesis, $\texttt{Level}_{\texttt{MIS}}(q)$ is even if and only if $q.dominator$, for all q such that $\texttt{Level}_{\texttt{MIS}}(q) = L - 1$.

Note that $\texttt{Level}_{\texttt{MIS}}(p.parent) = L - 1$. By Macro $Parent(p)$, $p.parent.dominator \neq p.dominator$. Since L is even if and only if $L - 1$ is not even, we are done. $\square$

We can require that $\mathcal{BFST}$ stabilize in $O(n)$ rounds and use $O(\log n)$ space per process [16, 8]. By Theorem 1, Lemmas 2 and 6, we have:

Theorem 2 *$\mathcal{MIST} \circ \mathcal{BFST}$ is a silent self-stabilizing algorithm that builds an MIS Tree within $O(n)$ rounds using $O(\log n)$ space per process.*

Height of the MIS Tree The next property establishes a bound on the height of the MIS Tree computed by $\mathcal{MIST} \circ \mathcal{BFST}$. We then illustrate this property with an example matching the bound.

Lemma 7 *In any terminal configuration of $\mathcal{MIST} \circ \mathcal{BFST}$, if p is a non-root process at even level of T_{MIS}, then the process $p.parent$ is at level $\texttt{Level}_{\texttt{BFS}}(p) - 1$ in T_{BFS}.*

Proof. As p is a dominator process, no one of its neighbors is dominator by Lemma 3. Since p is not the root, $\texttt{Parent}_{\texttt{BFS}}(p)$ is defined. To sum up, $\texttt{Parent}_{\texttt{BFS}}(p) \in \mathcal{N}_p$ and $\texttt{Level}_{\texttt{BFS}}(\texttt{Parent}_{\texttt{BFS}}(p)) = \texttt{Level}_{\texttt{BFS}}(p) - 1$, so $\min\{\texttt{Level}_{\texttt{BFS}}(q) \mid q \in \mathcal{N}_p \wedge q.dominator \neq p.dominator\} = \texttt{Level}_{\texttt{BFS}}(p) - 1$. By definition, for all q, $\texttt{Level}_{\texttt{BFS}}(q) < \texttt{Level}_{\texttt{BFS}}(p)$ implies $Priority(q) < Priority(p)$. By Macro $Parent(p)$, we are done. $\square$

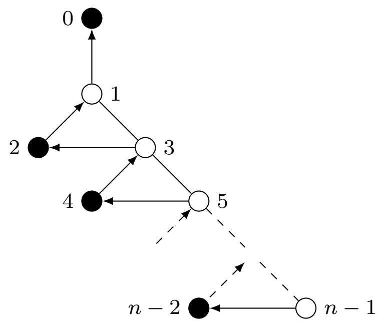


Figure 2: Worst case example for MIS tree height.

Property 2 *In any terminal configuration of $\mathcal{MIST} \circ \mathcal{BFST}$, the height of the MIS tree T_{MIS} of G computed by $\mathcal{MIST} \circ \mathcal{BFST}$ is at most $2 \times \mathcal{D}$, where $\mathcal{D}$ is the diameter of G.*

Proof. Let H be the height of T_{MIS}. Let $\sigma = (p_\ell, p_{\ell-1}, \ldots, p_0 = r)$ be any path in T_{MIS} from a leaf to the root. That is, p_ℓ is a leaf, and $p_j = p_{j+1}.parent$ for all $j < \ell$.

Since T_{MIS} is 2-colored *w.r.t.* $dominator$ variables, any path in T_{MIS} is also 2-colored *w.r.t.* $dominator$ variables. Moreover, $p_0.dominator = true$, so $p_j.dominator \equiv (j\%2 = 0)$, for all $j < \ell$.

Since $Priority(p_{j+1}) > Priority(p_j)$ (Lemma 4), we have:

(a) $\texttt{Level}_{\texttt{BFS}}(p_{j+1}) \geq \texttt{Level}_{\texttt{BFS}}(p_j)$ for all $j < \ell$.

By Lemma 7, $\texttt{Level}_{\texttt{BFS}}(p.parent) < \texttt{Level}_{\texttt{BFS}}(p)$ for any dominator process $p \neq r$. Thus:

(b) For all j, if j is odd, then $\texttt{Level}_{\texttt{BFS}}(p_{j+1}) > \texttt{Level}_{\texttt{BFS}}(p_j)$.

From (a) and (b), it follows that:

(c) At most two processes of σ can be on any one level of T_{BFS}.

By definition of T_{BFS}:

(d) $p_0 = r$ is the only process of σ at level 0 in T_{BFS}.

By definition of T_{BFS} and (d), p_1 (if defined) is at level 1 in both T_{BFS} and T_{MIS}. Then, by (b), p_2 (if defined) is not at the same level in T_{BFS} as p_1. So, p_0 and p_2 are not at the same level as p_1 in T_{BFS}, that is:

(e) p_1 is the only process of σ at level 1 in T_{BFS}.

Hence, among the $\ell + 1$ processes of σ, there are exactly one process at level zero of T_{BFS}, one process at level 1 of T_{BFS}, and for every other level x of T_{BFS}, there are at most two processes of σ at level x by (c). Hence, $\ell \leq 2 \times (H-1) + 2$, that is, $\ell \leq 2 \times H \leq 2 \times \mathcal{D}$. □

Figure 2 exhibits the upper bound on the height of T_{MIS}, depending on the diameter $\mathcal{D}$ of the network. Even processes have the same parent in both T_{BFS} and T_{MIS}, whereas odd ones have their parent in T_{MIS} at the same level in T_{BFS}. It is not possible to increase the height of T_{MIS} more than once per level of T_{BFS}, thus the height of T_{MIS} is at most twice the one of T_{BFS}, that is $2 \times \mathcal{D}$.

4 k-Clustering of at most $1 + \left\lfloor \frac{n-1}{k+1} \right\rfloor$ k-clusters

In this section, we present a silent self-stabilizing algorithm, called $\mathcal{CLR}(k)$, which constructs a k-clustering of at most $1 + \left\lfloor \frac{n-1}{k+1} \right\rfloor$ distinct k-clusters in a directed tree network. Its stabilization time is $O(H)$ rounds, where H is the height of the tree. By composing $\mathcal{CLR}(k)$ with any silent self-stabilizing spanning tree algorithm, we obtain a silent self-stabilizing k-clustering algorithm that builds at most $1 + \left\lfloor \frac{n-1}{k+1} \right\rfloor$ distinct k-clusters in any arbitrary network. Moreover, we will see in Section 5 that $\mathcal{CLR}(k) \circ \mathcal{MIST} \circ \mathcal{BFST}$ is a silent self-stabilizing k-clustering algorithm which is $7.2552k + O(1)$-competitive in any UDG network. The stabilization time of $\mathcal{CLR}(k) \circ \mathcal{MIST} \circ \mathcal{BFST}$ is $O(n)$ rounds and its memory requirement is $O(\log n)$ space per process.

4.1 Algorithm $\mathcal{CLR}(k)$

We assume that the network is a rooted tree T with root r.

The formal description of $\mathcal{CLR}(k)$ is given in Algorithm 2. $\mathcal{CLR}(k)$ builds a k-clustering in two phases. During the first phase, $\mathcal{CLR}(k)$ computes the set of clusterheads, Dom, which has cardinality at most $1 + \left\lfloor \frac{n-1}{k+1} \right\rfloor$. The second phase consists of building a spanning forest, where each directed tree is rooted at a clusterhead and represents the k-cluster of that clusterhead. Hence, we obtain a k-clustering of at most $1 + \left\lfloor \frac{n-1}{k+1} \right\rfloor$ k-clusters. $\mathcal{CLR}(k)$ uses the following three variables in the code of each process p:

- $p.\alpha$, an integer in the range $[0..2k]$. In any terminal configuration, the set of clusterheads Dom is defined as the set of processes p such that $p.\alpha = k$ or $p.\alpha < k$ and $p = r$.
- $p.parent_{CLR} \in \mathcal{N}_p$. In any terminal configuration, $p.parent_{CLR}$ is the parent of p in its k-cluster, unless p is a clusterhead, in which case $p.parent_{CLR} = p$.
- $p.head_{CLR}$. In any terminal configuration, $p.head_{CLR}$ is equal to the identifier of the clusterhead in the k-cluster that p belongs to.

Algorithm 2 $\mathcal{CLR}(k)$, code for each process p

```
Inputs: $\texttt{Parent}(p) \in \mathcal{N}_p$
Variables: $p.\alpha \in [0..2k]$ ; $p.parent_{CLR} \in \mathcal{N}_p \cup \{p\}$ ; $p.head_{CLR} \in V$
Macros:
  $IsShort(p)$           $\equiv$  $p.\alpha < k$
  $IsTall(p)$            $\equiv$  $p.\alpha \geq k$
  $IsClusterHead(p)$     $\equiv$  $(p.\alpha = k) \vee (IsShort(p) \wedge (p = r))$
  $ShortChildren(p)$     $=$  $\{q \mid (\texttt{Parent}(q) = p) \wedge IsShort(q)\}$
  $TallChildren(p)$      $=$  $\{q \mid (\texttt{Parent}(q) = p) \wedge IsTall(q)\}$
  $MaxAShort(p)$         $=$  if $ShortChildren(p) = \emptyset$ then $-1$ else $\max\{q.\alpha \mid q \in ShortChildren(p)\}$
  $MinATall(p)$          $=$  if $TallChildren(p) = \emptyset$ then $2k+1$ else $\min\{q.\alpha \mid q \in TallChildren(p)\}$
  $MinIDMinATall(p)$     $=$  if $TallChildren(p) = \emptyset$ then $p$ else $\min\{q \in TallChildren(p) \mid q.\alpha = MinATall(p)\}$
  $Alpha(p)$             $=$  if $MaxAShort(p) + MinATall(p) \leq 2k-2$ then $MinATall(p)+1$ else $MaxAShort(p)+1$
  $Parent_{CLR}(p)$      $=$  if $p.\alpha < k$ then $\texttt{Parent}(p)$ else if $p.\alpha = k$ then $p$ else $MinIDMinATall(p)$
  $Head_{CLR}(p)$        $=$  if $IsClusterHead(p)$ then $p$ else $p.parent_{CLR}.head_{CLR}$
Actions:
  SetAlpha   ::  $p.\alpha \neq Alpha(p)$                  $\longrightarrow$  $p.\alpha \leftarrow Alpha(p)$
  SetParent  ::  $p.parent_{CLR} \neq Parent_{CLR}(p)$     $\longrightarrow$  $p.parent_{CLR} \leftarrow Parent_{CLR}(p)$
  SetHead    ::  $p.head_{CLR} \neq Head_{CLR}(p)$         $\longrightarrow$  $p.head_{CLR} \leftarrow Head_{CLR}(p)$
```

Building Dom The first phase of $\mathcal{CLR}(k)$ consists of building the set Dom as a k-dominating set of T, that is, a subset of processes such that every process is at most at distance k from a process in Dom. Dom is constructed by dynamic programming, starting from the leaves of T. As previously explained, Dom is defined using the values of $p.\alpha$ for all p.

Consider any terminal configuration. In this configuration, $p.\alpha = \|p, q\|$, where q is the furthest process in the subtree of T rooted at p, that will be in the same k-cluster as p.

- If $p.\alpha < k$, then p is said to be *short* and we have two cases: $p \neq r$ or $p = r$. In the former case, p is k-dominated by a process of Dom outside of its subtree, that is, the path from p to its clusterhead goes through the parent link of p in the tree, and the distance to this process is at most $k - p.\alpha$. In the latter case, p is not k-dominated by any other process of Dom inside its subtree and, by definition, there is no process outside its subtree. Thus, p must be placed in Dom.
- If $p.\alpha \geq k$, then p is said to be *tall* and there is a process q at $p.\alpha - k$ hops below p such that $q.\alpha = k$. So, $q \in Dom$ and p is k-dominated by q. Note that, if $p.\alpha = k$, then $p.\alpha - k = 0$, that is, $p = q$ and p belongs to Dom.

$p.\alpha$ is computed using the two following macros:

- $MaxAShort(p)$ returns the maximum value of $q.\alpha$ for all *short* children q of p. If p has no *short* children, $MaxAShort(p)$ returns -1.
- $MinATall(p)$ returns the minimum value of $q.\alpha$ for all *tall* children q of p. If p has no *tall* children, $MinATall(p)$ returns $2k + 1$.

According to these macros, $p.\alpha$ is computed by Action SetAlpha in a bottom-up fashion as follows:

- If p is a leaf, $p.\alpha = 0$.
- If p is not a leaf and $MaxAShort(p) + MinATall(p) \leq 2k - 2$, $p.\alpha = MinATall(p) + 1$.
- If p is not a leaf and $MaxAShort(p) + MinATall(p) > 2k - 2$, $p.\alpha = MaxAShort(p) + 1$.

To help the reader's intuition, we summarize below the important properties of $p.\alpha$, for any process p. These properties can be checked in the examples presented in Figure 3.

Property 3 *In any terminal configuration, for every process p, we have:*

(a) If $p.\alpha > 0$, then there is some child q of p such that $q.\alpha = p.\alpha - 1$.

(b) If $p.\alpha > k$, then there is a proper descendant q of p such that $q \in Dom$ and q is $p.\alpha - k$ levels below p.

(c) There is a member of Dom within $|p.\alpha - k|$ hops of p.

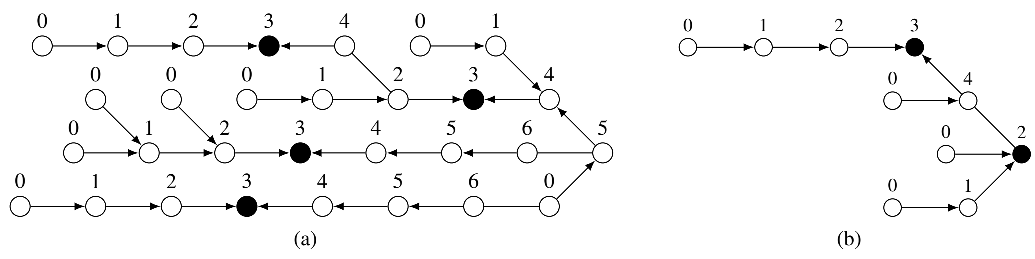


Figure 3: Examples of 3-Clustering using $\mathcal{CLR}(3)$. The root of each tree network is on the right, values of α are indicated, clusterheads are colored in black, and arrows represent local spanning tree of each k-cluster.

Constructing k-Clustering The second phase of $\mathcal{CLR}(k)$ partitions the processes into distinct k-clusters, each of which contains one clusterhead. Each k-cluster contains a *k-cluster spanning tree*, a tree containing all the processes of that k-cluster. Each k-cluster spanning tree is a subgraph of T rooted at the clusterhead, possibly with the directions of some edges reversed. Furthermore, the height of the k-cluster spanning tree is at most k.

Each process of Dom designates itself as clusterhead using Actions SetParent and SetHead. Other processes p designate their parent (using Action SetParent) as follows: (1) if p is *short*, then its parent in its k-cluster is its parent in the tree; (2) if p is *tall*, then p selects as parent in its k-clustering its *tall* child in the tree of minimum α value. Finally, identifiers of clusterheads are propagated in a top-down fashion in their k-cluster using Action SetHead.

Two examples of 3-clustering using $\mathcal{CLR}(3)$ are given in Figure 3. In Figure 3a, the root is a *tall* process, consequently it is not a clusterhead. In Figure 3b, the root is a *short* process, consequently it is a clusterhead.

4.2 Correctness

We first show the convergence of $\mathcal{CLR}(k)$ from any configuration to a terminal one. Since computation of the $p.\alpha$ is bottom-up in T, the time required for those values to stabilize is $O(H)$ rounds. After that, one additional round is necessary to fix the *Parent*$_{CLR}$ variables, because the values of these variables only depend on the α variables. Finally, the *head*$_{CLR}$ variables are fixed top-down within the k-cluster spanning trees starting from the clusterheads in $O(H)$ rounds. Hence, it follows that the time complexity of $\mathcal{CLR}(k)$ is $O(H)$ rounds, as shown below.

Lemma 8 *For every process p, the variable $p.\alpha$ is fixed forever within $H+1$ rounds.*

Proof. We prove this lemma by backwards induction on the level $\texttt{Level}(p)$ of processes p in the tree.

As a base case, if $\texttt{Level}(p) = H$, that is p is a leaf, then $p.\alpha$ is fixed forever within one round.

Assume for every p such that $\texttt{Level}(p) = l$, the variable $p.\alpha$ is fixed forever within $H - l + 1$ rounds.

Let q be a process such that $\texttt{Level}(q) = l - 1$. The value of *Alpha*$(q)$ depends only on the values of every $p.\alpha$ where p has level l. By the induction hypothesis, all those values are fixed within $H - l + 1$ rounds, thus $q.\alpha$ is fixed within one additional round, that is within $H - l + 2 = H - (l-1) + 1$ rounds.

This complexity is maximum with $l = 0$ and the lemma follows. □

Lemma 9 *For every process p, the variable $p.$*parent$_{CLR}$ *is fixed forever within $H+2$ rounds.*

Proof. As the evaluation of both guard and statement of Action $SetParent$ only relies, for a process p, on the variables $p.parent_{CLR}$ and $q.\alpha$ for every q neighbor of p. Thus, after all α variables are fixed in the network, every $p.parent_{CLR}$ is fixed within one additional round. By Lemma 8, we are done. □

Lemma 10 *In every configuration where all* parent$_{CLR}$ *and α variables are fixed forever, there is no directed cycle constituted of directed edges of the form $(p, p.$*parent$_{CLR}$*) except self-loops.*

Proof. The network being a tree, we only need to exclude the existence of cycle of size two. Assume by the contradiction that such a cycle exists between p and its neighbor q, that is $p.parent_{CLR} = q$ and $q.parent_{CLR} = p$. Without loss of generality, assume that q is a child of p. Then, by definition of Macro $Parent_{CLR}(q)$, $q.\alpha < k$. By definition of Macro $Parent_{CLR}(p)$, $q.\alpha > k$, a contradiction. The cluster level of the parent of each process p which is not a clusterhead is smaller than the cluster level of p, and thus no cycle of cluster parent pointers is possible. □

Lemma 11 *For every process p, the variable p.head$_{CLR}$ is fixed forever within $O(H)$ rounds.*

Proof. By Lemmas 8 and 9, the variables $p.\alpha$ and $p.parent_{CLR}$ are fixed within $H + 2$ rounds.

For every process p, the variable $p.head_{CLR}$ only depends on $p.parent_{CLR}.head_{CLR}$ and some fixed variables.

For every process p such that $p.parent_{CLR} = p$, $p.head_{CLR}$ is fixed forever in at most one additional round. Then, changes on $head_{CLR}$ can be propagated from node p to its neighbor q only if $q.parent_{CLR} = p$. By Lemma 10, these propagations end after $O(H)$ rounds, and we are done. □

From Lemmas 8 to 11, follows:

Lemma 12 *Starting from any configuration, $\mathcal{CLR}(k)$ reaches a terminal configuration in $O(H)$ rounds.*

We then consider any terminal configuration to show the closure of $\mathcal{CLR}(k)$. The proof begins by formally establishing the three claims given in Property 3, in Remark 2, Lemmas 13, and 14.

Remark 2 *Property 3.(a) follows immediately from the definition of α.*

Below, we prove Property 3.(b).

Lemma 13 *In any terminal configuration of $\mathcal{CLR}(k)$, for every process p, if $p.\alpha > k$, then there is a proper descendant q of p such that $q \in Dom$ and q is $p.\alpha - k$ levels below p.*

Proof. We prove this lemma by strong induction on $p.\alpha$.

As a base case, if $p.\alpha = k + 1$, then, by Property 3.(a), there is a child q of p such that $q.\alpha = k$, that is $q \in Dom$.

Assume the lemma holds for every p such that $k < p.\alpha < a$.

Let p' be a process such that $p'.\alpha = a$.

By Property 3.(a), there is a child q' of p' such that $q'.\alpha = p'.\alpha - 1$. By induction hypothesis, there is a proper descendant q'' of q' such that $q'' \in Dom$ and q'' is $q'.\alpha - k$ levels below q'. So, q'' is $q'.\alpha - k + 1 = p'.\alpha - 1 - k + 1 = p'.\alpha - k$ below p', and we are done. □

We now prove Property 3.(c).

Lemma 14 *In any terminal configuration of $\mathcal{CLR}(k)$, for every process p, there is a process q such that $q \in Dom$ and $\|p, q\| \leq |p.\alpha - k|$.*

Proof. If $p.\alpha > k$, then, by Lemma 13, we are done.

Consider now any process p such that $p.\alpha \leq k$. We prove the lemma by strong backwards induction on $p.\alpha$.

As a base case, if $p.\alpha = k$, then $p \in Dom$ by definition.

Assume the lemma holds for every p' such that $a < p'.\alpha \leq k$.

Let q be a process such that $q.\alpha = a$ and $q \neq r$. Indeed, if $r.\alpha \leq k$, then $r \in Dom$ by definition. Let q' be the parent of q. We consider two cases.

- Assume $q'.\alpha = MaxAShort(q') + 1$. As $q.\alpha < k$, q is *short* and $q.\alpha \leq MaxAShort(q')$. So:

$$\begin{aligned} q.\alpha &< q'.\alpha \leq k \\ a &< q'.\alpha \leq k \end{aligned}$$

By induction hypothesis, there is a member of Dom which is within $k - q'.\alpha$ hops of q'. Then, this process is within $k - q'.\alpha + 1$ hops from q. Now:

$$\begin{array}{ll} a & < q'.\alpha \\ -q'.\alpha & < -a \\ k - q'.\alpha + 1 & < k - a + 1 \\ k - q'.\alpha + 1 & \leq k - a \\ k - q'.\alpha + 1 & \leq k - q.\alpha \\ k - q'.\alpha + 1 & \leq |q.\alpha - k| \end{array}$$

So, this process is within $|q.\alpha - k|$ hops from q and we are done.

- Otherwise, $q'.\alpha = \mathit{MinATall}(q') + 1$ and $q'.\alpha > k$. By Lemma 13, there is some $q'' \in Dom$ within $q'.\alpha - k$ hops of q'. Thus, $\|q'', q\| \leq q'.\alpha - k + 1$. Then, by definition of α:

$$\begin{array}{ll} \mathit{MaxAShort}(q') + \mathit{MinATall}(q') & \leq 2k - 2 \\ \mathit{MinATall}(q') - k + 2 & \leq k - \mathit{MaxAShort}(q') \\ q'.\alpha - k + 1 & \leq k - q.\alpha \end{array}$$

Hence:

$$\begin{array}{ll} \|q'', q\| & \leq k - q.\alpha \\ \|q'', q\| & \leq |q.\alpha - k| \end{array}$$

So, q'' is within $|q.\alpha - k|$ hops from q and we are done.

□

Since $|p.\alpha - k| \leq k$ for every p, we can deduce the following corollary from Property 3.(c).

Corollary 1 *In any terminal configuration of $\mathcal{CLR}(k)$, Dom is a k-dominating set of T.*

The following lemma shows that every process is in the k-cluster of a member of Dom.

Lemma 15 *In any terminal configuration of $\mathcal{CLR}(k)$, for every process p, there is a path $P = (p_1 = p, \dots, p_m)$ such that: (1) $m \leq k$, (2) $\forall i \in [1..m-1], p_i.\text{parent}_{CLR} = p_{i+1}$, (3) $p_m.\text{parent}_{CLR} = p_m$, (4) $\forall i \in [1..m], p_i.\text{head}_{CLR} = p_m$, (5) $p_m \in Dom$.*

Proof. We prove this lemma by strong induction on $|p.\alpha - k|$. Note that $p.\alpha \in [0..2k]$, thus $|p.\alpha - k| \in [0..k]$.

As a base case, if $p.\alpha = k$, then $IsClusterHead(p) = \mathit{true}$. Thus, by definition, $p.\mathit{parent}_{CLR} = p$ and $p.\mathit{head}_{CLR} = p$. The path $P = (p)$ verifies each property stated in the lemma.

Assume the lemma holds for every q such that $|q.\alpha - k| < a$.

Let p be a process such that $|p.\alpha - k| = a$.

If $p.\alpha > k$, then, by definition of $Alpha(p)$, $p.\alpha = \mathit{MinATall}(p) + 1$, *i.e.*, there is some neighbor q of p such that $q.\alpha = \mathit{MinATall}(p)$. Without loss of generality, consider the one of smallest identifier, hence $p.\alpha = q.\alpha + 1$. Since $p.\alpha - k = a$, follows $q.\alpha + 1 - k = a$, that is $q.\alpha - k = a - 1 < a$. By induction hypothesis, there is a path $Q = (p_1 = q, \dots, p_m)$ leading to a clusterhead p_m such that:

- $m \leq k$,
- $\forall i \in [1..m-1], p_i.\mathit{parent}_{CLR} = p_i + 1$,
- $p_m.\mathit{parent}_{CLR} = p_m$,
- $\forall i \in [1..m], p_i.\mathit{head}_{CLR} = p_m$.

By definition of $Parent_{CLR}(p)$ and $Head_{CLR}(p)$, $p.parent_{CLR} = q$ and $p.head_{CLR} = p_m$, and the lemma holds.

Otherwise, $p.\alpha < k$. If $p = r$, then $IsClusterHead(p) = \mathit{true}$ and the lemma holds. Consider now the case $p \neq r$ and note $q = \texttt{Parent}(p)$. By definition of $Parent_{CLR}(p)$, $p.parent_{CLR} = q$. By definition of $Head_{CLR}(p)$, $p.headCLR = q.head_{CLR}$. We now show that $|q.\alpha - k| < a$, *i.e.*, $|q.\alpha - k| < |p.\alpha - k|$ in order to make use of the induction hypothesis as in the previous case, thus completing the proof. Two cases have to be distinguished:

- $q.\alpha \leq k$, then, by definition of $Alpha(q)$, $q.\alpha = \mathit{MaxAShort}(q) + 1$. As p is a *short* child of q, $q.\alpha \geq p.\alpha + 1$, and $q.\alpha - k > p.\alpha - k$. Since p and q are *short* processes, $|q.\alpha - k| < |p.\alpha - k|$.
- $q.\alpha > k$, then, by definition of $Alpha(q)$, $q.\alpha = \mathit{MinATall}(q) + 1$ and:

$$\begin{array}{lll}
\mathit{MaxAShort}(q) + \mathit{MinATall}(q) & \leq 2k - 2 & \\
(\mathit{MaxAShort}(q) + 1) + (q.\alpha - k) & \leq k & \\
(p.\alpha + 1) + (q.\alpha - k) & \leq k & (p.\alpha \leq \mathit{MaxAShort}(q)) \\
q.\alpha - k & \leq k - p.\alpha - 1 & \\
|q.\alpha - k| & < |k - p.\alpha| & \\
|q.\alpha - k| & < |p.\alpha - k| &
\end{array}$$

□

Lemma 16 *In any terminal configuration of $\mathcal{CLR}(k)$, every k-cluster whose clusterhead is not the root contains at least a path of $k+1$ processes.*

Proof. Consider any k-cluster whose clusterhead p is not the root. Then, $p.\alpha = k$, $p.parent_{CLR} = p$, and $p.head_{CLR} = p$ by definition of $\mathit{IsClusterHead}(p)$, $\mathit{Parent}_{CLR}(p)$, and $\mathit{Head}_{CLR}(p)$. Moreover, by Property 3.(a), there is a path $(p_0, \ldots, p_k)$ such that $p_k = p$ and for every $i \in [0..k-1]$, $p_i.\alpha = p_{i+1}.\alpha - 1 = i$. By Definition of Macro $\mathit{Parent}_{CLR}(p_j)$, for every $j \in [0..k-1]$, $p_j.parent_{CLR} = p_{j+1}$. By Definition of Macro $\mathit{Head}_{CLR}(p_j)$, for every $j \in [0..k-1]$, $p_j.head_{CLR} = p_{j+1}.head_{CLR} = p_k = p$. □

Lemma 17 *In any terminal configuration of $\mathcal{CLR}(k)$, there are at most $1 + \left\lfloor \frac{n-1}{k+1} \right\rfloor$ distinct k-clusters.*

Proof. By Lemma 16, except for the k-cluster which contains the root , every k-cluster contains at least $k+1$ processes. Thus, there are at most $1 + \left\lfloor \frac{n-1}{k+1} \right\rfloor$ k-clusters. □

By Corollary 1 and Lemmas 15 and 17, we have:

Lemma 18 *In any terminal configuration of $\mathcal{CLR}(k)$, T is partitioned into at most $1 + \left\lfloor \frac{n-1}{k+1} \right\rfloor$ distinct k-clusters.*

From Lemmas 12 and 18, we have:

Theorem 3 *In any tree of n processes and height H, $\mathcal{CLR}(k)$ is a silent self-stabilizing algorithm that partitions the tree within $O(H)$ rounds into at most $1 + \left\lfloor \frac{n-1}{k+1} \right\rfloor$ distinct k-clusters.*

By Theorems 1, 2, and 3, $\mathcal{CLR}(k) \circ \mathcal{MIST} \circ \mathcal{BFST}$ is self-stabilizing, $\mathcal{MIST} \circ \mathcal{BFST}$ stabilizes within $O(n)$ rounds, and $O(H)$ rounds later $\mathcal{CLR}(k) \circ \mathcal{MIST} \circ \mathcal{BFST}$ reaches a terminal configuration, where H is the height of T_{MIS}. Now, by Property 2 (page 7), H is bound by $2\mathcal{D}$, where $\mathcal{D}$ is the diameter of the network. Hence, from any initial configuration, $\mathcal{CLR}(k) \circ \mathcal{MIST} \circ \mathcal{BFST}$ stabilizes in $O(n)$ rounds.

Theorem 4 *In any arbitrary network with unique IDs, $\mathcal{CLR}(k) \circ \mathcal{MIST} \circ \mathcal{BFST}$ is a silent self-stabilizing algorithm that builds at most $1 + \left\lfloor \frac{n-1}{k+1} \right\rfloor$ distinct k-clusters within $O(n)$ rounds using $O(\log n)$ space per process.*

5 Competitiveness of k-Clustering

Unit Disk Graphs We now analyze the competitiveness, in terms of number of clusters, of $\mathcal{CLR}(k) \circ \mathcal{MIST} \circ \mathcal{BFST}$, in the special case that the network is a UDG in the plane, that is, the processes are fixed in the plane, and two processes can communicate if and only if their Euclidean distance in the plane is at most one. We first show, in Lemma 19, that the cardinality of the MIS computed by $\mathcal{MIST} \circ \mathcal{BFST}$ is bounded by a constant multiple of the minimum cardinality of any k-clustering, then in Lemma 20, we show that the cardinality of Clr, the k-clustering built by $\mathcal{CLR}(k) \circ \mathcal{MIST} \circ \mathcal{BFST}$, is bounded by a constant multiple of that same minimum.

Lemma 19 *For every connected UDG and every $k \geq 1$, any independent set I is of cardinality at most $\left(\frac{2\pi k^2}{\sqrt{3}} + \pi k + 1\right)$ times the cardinality of an optimum k-clustering Opt.*

Proof. We make use of a result by Folkman and Graham [14]. If X is a compact convex region of the plane, let $J \subseteq X$ such that the distance between any two distinct members of J is at least 1. Then, the cardinality of J is at most $\left\lfloor \frac{2}{\sqrt{3}}A(X) + \frac{1}{2}P(X) + 1 \right\rfloor$, where $A(X)$ and $P(X)$ are the area and the perimeter of X, respectively. We observe that J is any independent set of any UDG in the plane. Consider any clusterhead p in Opt and the surrounding disk of radius k centered at p in the plane. All processes that belongs to the k-cluster of p are within this disk. Due to the above result, no more than $\left(\frac{2}{\sqrt{3}}(\pi k^2) + \frac{1}{2}(2\pi k) + 1\right)$ processes can be independent in this disk, thus in the k-cluster of p. By definition, every process belongs to a k-cluster. It follows that the cardinality of any independent set is at most $\left(\frac{2\pi k^2}{\sqrt{3}} + \pi k + 1\right)$ times the one of an optimum k-clustering Opt. □

We now compare the maximal independent set computed by $\mathcal{MIST} \circ \mathcal{BFST}$ with the k-clustering set Clr computed by $\mathcal{CLR}(k) \circ \mathcal{MIST} \circ \mathcal{BFST}$.

Lemma 20 *For every connected network and every $k \geq 1$, let I be the MIS computed by $\mathcal{MIST} \circ \mathcal{BFST}$, the cardinality of Clr, the k-clustering built by $\mathcal{CLR}(k) \circ \mathcal{MIST} \circ \mathcal{BFST}$ is at most $1 + \frac{2}{k}(|I| - 1)$.*

Proof. By Lemma 16 (page 13), every k-cluster of Clr contains a path of $k+1$ processes (*i.e.*, of length k), excepted for the k-cluster which contains r. Since Clr is built on T_{MIS}, by Property 1 (page 4), this path contains $\lceil \frac{k}{2} \rceil$ processes of $I \setminus \{r\}$. Thus, $|Clr| - 1$ k-clusters of Clr contain at least $\lceil \frac{k}{2} \rceil$ processes of $I \setminus \{r\}$. We have:

$$\begin{array}{ll}
(|Clr| - 1) \times \lceil \frac{k}{2} \rceil & \leq |I \setminus \{r\}| \\
(|Clr| - 1)\frac{k}{2} & \leq |I| - 1 \\
|Clr| - 1 & \leq \frac{2}{k}(|I| - 1) \\
|Clr| & \leq 1 + \frac{2}{k}(|I| - 1)
\end{array}$$

□

By Lemmas 19 and 20, we deduce that $|Clr| \leq 1 + \left(\frac{4\pi k}{\sqrt{3}} + 2\pi\right)|Opt|$, and since $\frac{4\pi}{\sqrt{3}} \approx 7.2552$, we can claim the following:

Theorem 5 *For every connected UDG and every $k \geq 1$, $\mathcal{CLR}(k) \circ \mathcal{MIST} \circ \mathcal{BFST}$ computes a $7.2552k + O(1)$-approximation of the optimum k-clustering in terms of cardinality.*

Approximate Disk Graphs More generally, if V is a set of points in the plane, and $\lambda \geq 1$, then we say that $G = (V, E)$ is an *approximate disk graph* in the plane with *approximation ratio* λ, if, for any $u, v \in V$, $\|u, v\| \leq 1 \Rightarrow \{u, v\} \in E$ and $\{u, v\} \in E \Rightarrow \{u, v\} \leq \lambda$. This model has been first introduced by [2]. It is also known as Quasi-UDG from [17].

Theorem 6 *For every connected approximate disk graph in the plane with approximation ratio λ, and every $k \geq 1$, $\mathcal{CLR}(k) \circ \mathcal{MIST} \circ \mathcal{BFST}$ computes a $7.2552\lambda^2 k + O(\lambda)$-approximation of the optimum k-clustering in terms of cardinality.*

Proof. As in the proof of Lemma 19, we make use of the result of Folkman and Graham, but we then consider the surrounding disk of radius λk centered at any clusterhead of *Opt*. It follows that no more than $\left(\frac{2}{\sqrt{3}}(\pi\lambda^2k^2) + \frac{1}{2}(2\pi\lambda k) + 1\right)$ processes can be independent in this disk, and thus no more that that same number can be in any k-cluster of *Opt*. It follows that the cardinality of any independent set in an ADG is at most $\left(\frac{2\pi\lambda^2k^2}{\sqrt{3}} + \pi\lambda k + 1\right)$ times the one of an optimum k-clustering *Opt*. By Lemma 20 and since $\frac{4\pi}{\sqrt{3}} \approx 7.2552$, we are done. □

6 MIS Construction and Nick's Class

The time bottleneck of our k-clustering solution is the MIS Tree construction. Indeed, our algorithm builds a MIS Tree in $\Theta(n)$ rounds in the worst case (Theorem 2, page 7) and, once the MIS Tree is built, the k-clustering is computed in $O(\mathcal{D})$ rounds by Theorem 3 (page 13) and Property 2 (page 7). So, we would like to improve that time to be $O(\mathcal{D})$, but as we shall see below, finding an algorithm with a sublinear time complexity for computing an MIS tree for a general network could be very hard, and may be impossible.

Nick's Class ($\mathcal{NC}$) [4] is defined to be the set of all problems that can be solved in parallel in polylogarithmic time with polynomially many processors. Thus, there can be no deterministic polylogarithmic time distributed algorithm for any problem which is not in $\mathcal{NC}$. $\mathcal{P}$ is defined to be the set of all problems that can be deterministically solved in polynomial time. A problem $\mathbb{A} \in \mathcal{P}$ is said to be *$\mathcal{P}$-complete* if, given any problem $\mathbb{B} \in \mathcal{P}$, there is a reduction of $\mathbb{B}$ to $\mathbb{A}$, and that reduction can be computed in parallel in polylogarithmic time with polynomially many processors. Thus, $\mathcal{NC} = \mathcal{P}$ if and only if there is any one $\mathcal{P}$-complete problem which is in $\mathcal{NC}$.

The question of whether $\mathcal{NC} = \mathcal{P}$ is considered to be in the same class of difficulty as the question of whether $\mathcal{P} = \mathcal{NP}$. Just as we justify giving up the search for a polynomial time algorithm for any problem that we can prove to be $\mathcal{NP}$-complete, we justify giving up the search for a fast parallel algorithm for a problem if we can prove that it is $\mathcal{P}$-complete. We show that the exact problem solved by our MIS Tree construction is *$\mathcal{P}$-complete*.

Given a network $G = (V, E)$, we compute an MIS of G, with respect to priorities ordering defined in Section 3. Note that there is a natural lexical ordering on the subsets of V, obtained by writing each subset as an ordered list of processes. The MIS computed by our algorithm comes first in the natural lexical ordering (*w.r.t.* the priorities) of subsets of V, it is said to be the *lexically first maximal independent set* of G.

Let denote by $p_0, p_1, \dots, p_n$ the processes of G, ordered by priority. Our algorithm takes advantage of an additional property of priorities: There is a unique local minimum, *i.e.*, for any $i > 0$ there is some $j < i$ such that p_j is a neighbor of p_i.

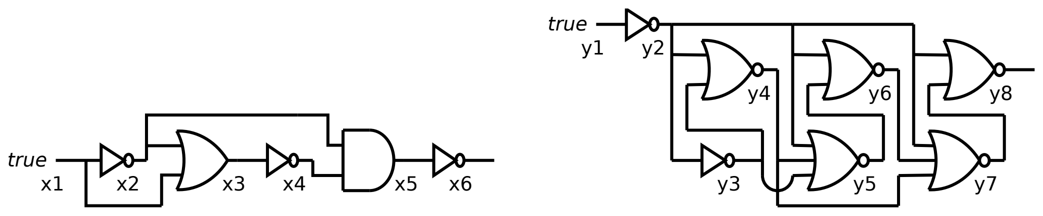
The lexically first maximal independent set problem on a graph G is equivalent to finding a lexically first maximal clique in the complementary graph G', shown by Cook [5] to be $\mathcal{P}$-complete.

However, our algorithm solves a restricted version of the LFMIS problem, where the ordering is known to have a unique local minimum, and thus we need to give separate proof that this version is also $\mathcal{P}$-complete. The below proof consists of exhibiting a method to $\mathcal{NC}$-reduce any instance of the $\mathcal{P}$-complete *Circuit Value* problem to an instance of the LFMIS problem with unique local minimum. The *Circuit Value* (CV) problem, is defined as the problem of evaluating the last output of an acyclic Boolean circuit, given that its inputs are assigned to *true*. Such a circuit consists of Boolean assignments (negation, conjunction or disjunction), inputs and outputs. This problem has been shown to be $\mathcal{P}$-complete in [18].

Theorem 7 *The LFMIS problem with unique local minimum is $\mathcal{P}$-complete.*

Proof. We prove this theorem by exhibiting a method to $\mathcal{NC}$-reduce any instance of the $\mathcal{P}$-complete CV problem to an instance of the LFMIS problem with unique local minimum. It goes through two transformation steps, first rewriting any instance of the CV problem into an intermediate constrained form, which can then be directly converted into an instance of the LFMIS problem.

First, consider an instance of CV problem. Denote by x_i, its i^{th} assigned variable. Without loss of generality, we assume that the number of variables is even, and that the i^{th} variable is assigned to $\neg x_{i-1}$ if i is even, and is assigned to either *true* or the conjunction or disjunction of two prior variables if i is odd.



(a) (b)

Figure 4: (a) An instance of CVP, and (b) its constrained form.

		1 :	$y_1 \leftarrow true$	
		2 :	$y_2 \leftarrow \neg y_1$	
1 :	$x_1 \leftarrow true$	3 :	$y_3 \leftarrow \neg y_2$	$x_1 \equiv y_3$
2 :	$x_2 \leftarrow \neg x_1$	4 :	$y_4 \leftarrow \neg y_2 \wedge \neg y_3$	$x_2 \equiv y_4$
3 :	$x_3 \leftarrow x_1 \vee x_2$	5 :	$y_5 \leftarrow \neg y_2 \wedge \neg y_3 \wedge \neg y_4$	$x_3 \equiv y_6$
4 :	$x_4 \leftarrow \neg x_3$	6 :	$y_6 \leftarrow \neg y_2 \wedge \neg y_5$	$x_4 \equiv y_5$
5 :	$x_5 \leftarrow x_2 \wedge x_4$	7 :	$y_7 \leftarrow \neg y_2 \wedge \neg y_4 \wedge \neg y_6$	$x_5 \equiv y_7$
6 :	$x_6 \leftarrow \neg x_5$	8 :	$y_8 \leftarrow \neg y_2 \wedge \neg y_7$	$x_6 \equiv y_8$
	(a)		(b)	(c)

Figure 5: (a) An instance of CVP, (b) its constrained form, and (c) variables correspondence.

The above assumptions can be enforced by using the following insertions of assignments, which can be done in parallel. In the case of an odd number of assignments, $a \leftarrow \neg x; b \leftarrow a \wedge a; c \leftarrow \neg b$, where x was the last variable and a, b, c are new variables. If an assignment at even rank is not a negation, then the same new assignments are inserted both ahead and behind that assignment. If an assignment at odd rank is a negation, then the assignment $a \leftarrow b \wedge b$ is inserted both ahead and behind that assignment.

Thus, assuming an even number of variables of the circuit, we denote them $x_1, x_2, \ldots, x_{2n}$. The first variable, x_1, is the first input, while the last variable, x_{2n}, is the last output of the circuit. We show an example of an instance of the CV problem in Figure 4a, and as a program in Figure 5a.

For any $1 \leq i \leq n$, we will refer to x_{2i-1} and x_{2i} as *partners*. Note that partners always take opposite Boolean values when evaluated.

Then, we rewrite that circuit in a constrained form, where variables are noted $y_1, y_2, \ldots, y_{2n+2}$. The first statement will be $y_1 \leftarrow$ *true*, and the second statement will be $y_2 \leftarrow \neg y_1$. For any $1 \leq i \leq n$, the two variables y_{2i+1} and y_{2i+2} will correspond to the partner variables x_{2i-1} and x_{2i}, in either order.[2] Thus $y_{2i+1} \equiv \neg y_{2i+2}$. We will also refer to y_{2i+1} and y_{2i+2} as partners. We use the following rewriting rules to construct the intermediate circuit, for any $1 \leq i \leq n$.

1. The $(2i+2)^{nd}$ assignment will be $y_{2i+2} \leftarrow \neg y_2 \wedge \neg y_{2i+1}$. That is, y_{2i+2} is assigned to the opposite Boolean value of its odd partner y_{2i+1}. Note that the term $\neg y_2$ does not impact the evaluation of the conjunction, since its value is *true*.

2. The $(2i+1)^{st}$ assignment will depend on the operator of the $(2i-1)^{st}$ assignment in the initial instance:

   (a) If the $(2i-1)^{st}$ assignment is an input *true*, then the $(2i+1)^{st}$ assignment will be $y_{2i+1} \leftarrow \neg y_2$, y_{2i+1} corresponds to x_{2i-1}, and y_{2i+2} corresponds to x_{2i}. Indeed, both x_{2i-1} and y_{2i+1} will be evaluated to *true*.

   (b) If the $(2i-1)^{st}$ assignment is a conjunction $x_{2i-1} \leftarrow x_j \wedge x_k$, let y_p and y_q be the variables corresponding to the partners of x_j and x_k, respectively. Then, the $(2i+1)^{st}$ assignment

[2] Actually, this order will depend on the rewriting rules explained next.

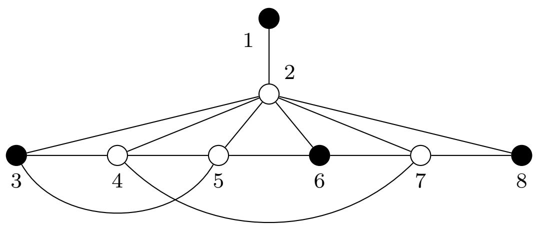


Figure 6: Resulting instance of the LFMIS problem.

will be $y_{2i+1} \leftarrow \neg y_2 \wedge \neg y_p \wedge \neg y_q$, y_{2i+1} corresponds to x_{2i-1}, and y_{2i+2} corresponds to x_{2i}. Indeed, the partners of x_j and x_k are evaluated to $\neg x_j$ and $\neg x_k$, and y_p and y_q will be evaluated similarly, thus there will be $y_{2i+1} \equiv \neg\neg x_j \wedge \neg\neg x_k \equiv x_j \wedge x_k \equiv x_{2i-1}$.

(c) If the $(2i-1)^{st}$ assignment is a disjunction $x_{2i-1} \leftarrow x_j \vee x_k$, let y_p and y_q be the variables corresponding to x_j and x_k, respectively. Then, the $(2i+1)^{st}$ assignment will be $y_{2i+1} \leftarrow \neg y_2 \wedge \neg y_p \wedge \neg y_q$, y_{2i+1} corresponds to x_{2i}, and y_{2i+2} corresponds to x_{2i-1}. Indeed, there will be $y_{2i+1} \equiv \neg x_j \wedge \neg x_k \equiv \neg(x_j \vee x_k) \equiv \neg x_{2i-1} \equiv x_{2i}$.

Through simple induction on sets of partner variables, we can see that evaluation of both circuits will assign to each variable of the intermediate circuit the same Boolean value as the corresponding variable in the initial circuit. We show the intermediate instance corresponding to the first example in Figure 4b, and as a program in Table 5b. The correspondence between variables of both instances is given in Table 5c.

Finally, we construct an instance of the LFMIS problem with unique local minima. Let G be the network whose processes are $p_1, p_2, \ldots, p_{2n+2}$, and where p_1 is the root. For each $1 \leq j < i \leq 2n+2$, p_i is adjacent to p_j if and only if the term $\neg y_j$ appears in the i^{th} assignment of the intermediate circuit. Figure 6 shows the resulting instance of the LFMIS problem for the circuit described in Figure 4b, and as a program in Table 5b.

The first variable y_1 is assigned to *true*; it is equivalent to having the root process p_1 in the LFMIS. The second variable y_2 is the only one to depend on y_1 and, for every $3 \leq i \leq 2n+2$, y_i depends on y_2; p_2 is the central process of G and the only one at level 1. Every other variable is the conjunction of the negations of some previous variables, it implies that, for every $3 \leq i \leq 2n+2$, local computation of the LFMIS at process p_i only relies on prior processes $p_2, \ldots, p_{i-1}$.

By simple induction on processes ordering, we see that $p_i \in I$ if and only if y_i is assigned the value *true* in the intermediate circuit, that is also in the initial circuit.

We note that all the steps of the reduction could be accomplished in parallel. Thus, any instance of the CV problem can be $\mathcal{NC}$-reduced to an instance of the LFMIS problem with unique local minimum. □

Although the problem is technically open, Theorem 7 justifies not seeking an $O(\mathcal{D})$ time algorithm for computing the LFMIS.

7 Perspectives

An immediate extension of this work would be to sharpen the competitiveness' analysis of our k-clustering in any UDG. Another possible extension is to try to find another competitive construction for a UDG which can be performed in sublinear time. We feel it is worth investigating if it is possible to design a self-stabilizing k-clustering that is competitive in any connected network.

References

[1] A. D. Amis, R. Prakash, D. Huynh, and T. Vuong. Max-Min D-Cluster Formation in Wireless Ad Hoc Networks. In *IEEE INFOCOM*, pages 32–41, 2000. 1

[2] L. Barrière, P. Fraigniaud, and L. Narayanan. Robust position-based routing in wireless ad hoc networks with unstable transmission ranges. In *DIALM*, pages 19–27. ACM, 2001. 5

[3] E. Caron, A. K. Datta, B. Depardon, and L. L. Larmore. A Self-Stabilizing k-Clustering Algorithm for Weighted Graphs. *JPDC*, 70(11):1159–1173, 2010. 1

[4] S. A. Cook. Deterministic CFL's are Accepted Simultaneously in Polynomial Time and Log Squared Space. In *STOC*, pages 338–345. ACM, 1979. 6

[5] Stephen A. Cook. A Taxonomy of Problems with Fast Parallel Algorithms. *Information and Control*, 64:2–22, March 1985. International Conference on Foundations of Computation Theory. 6

[6] A. K. Datta, S. Devismes, and L. L. Larmore. A Self-Stabilizing $O(n)$-Round k-Clustering Algorithm. In *SRDS*, pages 147–155, 2009. 1

[7] A. K. Datta, L. L. Larmore, and P. Vemula. A Self-Stabilizing $O(k)$-Time k-Clustering Algorithm. *The Computer Journal*, page bxn071, 2009. 1

[8] Ajoy K. Datta, Lawrence L. Larmore, and Priyanka Vemula. Self-Stabilizing Leader Election in Optimal Space. In *SSS*, pages 109–123, 2008. 3.2, 3.2

[9] E. W. Dijkstra. Self-Stabilizing Systems in Spite of Distributed Control. *Commun. ACM*, 17:643–644, 1974. 1, 2

[10] S. Dolev. *Self-Stabilization*. MIT Press, 2000. 2

[11] S. Dolev, M. G. Gouda, and M. Schneider. Memory Requirements for Silent Stabilization. In *PODC*, pages 27–34, 1996. 2

[12] S. Dolev, A. Israeli, and S. Moran. Uniform Dynamic Self-Stabilizing Leader Election. *IEEE Trans. Parallel Distrib. Syst.*, 8:424–440, 1997. 2

[13] Y. Fernandess and D. Malkhi. K-Clustering in Wireless Ad Hoc Networks. In *POMC 2002*, pages 31–37, 2002. 1, 1, 1, 3

[14] J. H. Folkman and R. L. Graham. An Inequality in the Geometry of Numbers. *Canad. Math. Bull.*, 12:745–752, 1969. 5

[15] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 1

[16] S. T. Huang and N. S. Chen. A Self-Stabilizing Algorithm for Constructing Breadth-First Trees. *Inf. Process. Lett.*, 41:109–117, 1992. 3.2, 3.2

[17] F. Kuhn, R. Wattenhofer, and A. Zollinger. Ad-hoc networks beyond unit disk graphs. In *DIALM-POMC*, pages 69–78. ACM, 2003. 5

[18] R. E. Ladner. The Circuit Value Problem is Log Space Complete for P. *SIGACT News*, 7:18–20, January 1975. 6

[19] V. Ravelomanana. Distributed k-Clustering Algorithms for Random Wireless Multihop Networks. In *ICN*, pages 109–116, 2005. 1

[20] M. A. Spohn and J. J. Garcia-Luna-Aceves. Bounded-Distance Multi-Clusterhead Formation in Wireless Ad Hoc Networks. *Ad Hoc Networks*, 5:504–530, 2004. 1

[21] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2nd edition, 2001. 2