Competitive Self-Stabilizing k-Clustering

Ajoy K. Datta, Lawrence L. Larmore
School of Computer Science, University of Nevada, Las Vegas, USA
Email: Firstname.Lastname@unlv.edu

Stéphane Devismes, Karel Heurtefeux, Yvan Rivierre
VERIMAG *Lab., Université Joseph Fourier, Grenoble, France*
Email: Firstname.Lastname@imag.fr

***Abstract*—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 $7.2552\lambda^2k+O(\lambda)$-*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 rest 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.**

I. 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 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 [13]. 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* [10] 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.*, [9], [7], [3]. The solution in [9] stabilizes in $O(k)$ rounds using $O(k \log n)$ space per process. The algorithm given in [7] stabilizes in $O(n)$ rounds using $O(\log n)$ space per process. The algorithm given in [3] stabilizes in $O(k \cdot n)$ rounds using $O(k \log n)$ space per process.

In [6], an asynchronous silent self-stabilizing algorithm that computes a *k-dominating* set of at most $\lfloor \frac{n}{k+1} \rfloor$ processes is given. 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 others join their nearest clusterhead. The k-dominating set construction given in [6] 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.*, [9], [7], [3], [6]) 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], [11], [17], [18]. Of those, only [11] deals with competitiveness. Moreover, they are all written in message-passing model. Deterministic solutions given in [1], [11] 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 [18] 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 [17] 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 [11] 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] To the best of our knowledge, there is no self-stabilizing competitive solution to the k-clustering problem.

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 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 [11]. 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 [11]. 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 VI.

Roadmap: In the next section, we present the model used throughout this paper. In Section III, we give our self-stabilizing MIS tree construction. In Section IV, we give our self-stabilizing k-clustering algorithm. In Section V, we analyze the competitiveness of our k-clustering algorithm in UDGs and ADGs. In Section VI, we show that the problem we solved in Section III is $\mathcal{P}$-*complete*. Finally, in Section VII, we give some perspectives. Due to space limitation, some proofs have been omitted. All proofs are available in the technical report online at:

http://www-verimag.imag.fr/TR/TR-2011-16.pdf

[1]Actually, in [11], 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.

II. 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 abuse of notation, we shall identify any process with its ID, whenever convenient.

We assume the *shared memory model* of computation [10] , 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 $e = \gamma_0\gamma_1 \ldots \gamma_i \ldots$ 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.

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*. 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'', and so forth.

Self-Stabilization and Silence: Let $\mathcal{A}$ be a distributed algorithm. Let P be any predicate on configurations of $\mathcal{A}$. $\mathcal{A}$ *self-stabilizes* to P if there exists a non-empty subset of configurations $\mathcal{S}$ such that:

1) Every configuration of $\mathcal{S}$ satisfies P. *(Correctness)*
2) Every step of $\mathcal{A}$ starting from a configuration of $\mathcal{S}$ leads to a configuration of $\mathcal{S}$. *(Closure)*
3) Every execution of $\mathcal{A}$, starting from any arbitrary configuration, contains a configuration of $\mathcal{S}$. *(Convergence)*

The configurations of $\mathcal{S}$ are called the *legitimate configurations*. Conversely, all other configurations are said *illegitimate*.

An algorithm is *silent* 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* [6] which is a variant of *collateral composition* [19]. When we collaterally compose two algorithms $\mathcal{A}$ and $\mathcal{B}$, they 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 [6] 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.*

III. The MIS Tree

In this section, we first recall the data structure *MIS tree* (for Maximal Independent Set tree), introduced in [11]. 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 [11].

A. 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 of $T(p)$. 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.*

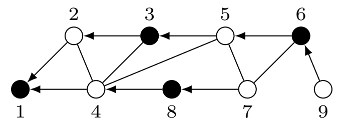

Figure 1: Example of LFMIST.

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 ith 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 encoding 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 $(1, 1, 2, 1, 3, 5, 8, 4, 6)$.

B. 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 [14] 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 either are 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 dominator. 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.

Correctness and Complexity Analysis: By Theorem 1, to show the correctness of $\mathcal{MIST} \circ \mathcal{BFST}$, we need only show that $\mathcal{MIST}$ constructs an 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.

Lemma 1 below shows 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, if at least $n + 1$ additional rounds have elapsed, no action of $\mathcal{MIST}$ is enabled.*

Proof Outline. Let γ be a configuration where no action of $\mathcal{BFST}$ is enabled. Starting from γ, $Priority(p)$ is fixed forever for every process p. Let $p_1, \ldots, p_n$ be the list of processes ordered by $\prec$ in γ.

The first part of the proof consists of showing by induction on the rank of every process in the ordering that all actions SetDominator are disabled forever after at most n rounds have elapsed.

Then, the values of $Priority(p)$ and *p.dominator* are fixed forever. For any processes, the guard of action SetParent depends only on those values. Thus, after at most one additional round, no action of $\mathcal{MIST}$ is enabled anymore, and we are done. $\square$

Let us 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. We deduce from the definition of $Dominator(p)$ that I is an MIS of G.

Consider then the subgraph T_{MIS} induced by the values of the parent pointers of $\mathcal{MIST}$ in γ. We show that T_{MIS} is a spanning tree of the network rooted at r. The proof is based on the following technical property: in γ, $Priority(p.parent) < Priority(p)$ for every process $p \neq r$.

Finally, r is at level zero of T_{MIS} and belong to I. By induction and using the definition of predicate $Parent(p)$, we show that a process is in I if and only if its level is even in T_{MIS}. In other words, T_{MIS} is an MIS tree of the network.

Hence, we can conclude with Lemma 2 that follows.

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  ::  $p.dominator \neq Dominator(p)$                                   ⟶  $p.dominator \leftarrow Dominator(p)$
  SetParent     ::  $p.dominator = Dominator(p) \wedge p.parent \neq Parent(p)$   ⟶  $p.parent \leftarrow Parent(p)$
```

Lemma 2 *In any configuration where no action of $\mathcal{MIST} \circ \mathcal{BFST}$ is enabled, T_{MIS} is an MIS tree of the network.*

We can require that $\mathcal{BFST}$ stabilizes in $O(n)$ rounds and use $O(\log n)$ space per process [14], [8]. By Theorem 1, Lemmas 1 and 2, 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 $\mathcal{MIST} \circ \mathcal{BFST}$. Below, we illustrate this property with an example matching the bound. (The proof is in the technical report.)

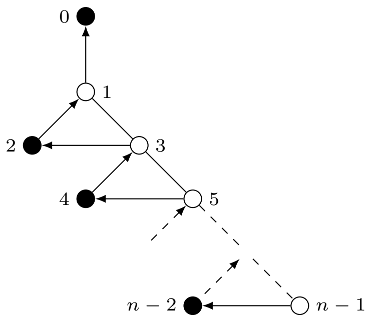

Figure 2: Worst case example for MIS tree height.

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

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 $2 \times \mathcal{D}$.

IV. 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 V 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.

A. Algorithm $\mathcal{CLR}(k)$

Assume that the network is a tree T rooted 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.

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.

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

```
Inputs: Parent(p) ∈ 𝒩_p ∪ {p}
Variables: p.α ∈ [0..2k] ; p.parent_CLR ∈ 𝒩_p ∪ {p} ; p.head_CLR ∈ V
Macros:
  IsShort(p)          ≡  p.α < k
  IsTall(p)           ≡  p.α ≥ k
  IsClusterHead(p)    ≡  (p.α = k) ∨ (IsShort(p) ∧ (p = r))
  ShortChildren(p)    =  {q ∈ 𝒩_p | (Parent(q) = p) ∧ IsShort(q)}
  TallChildren(p)     =  {q ∈ 𝒩_p | (Parent(q) = p) ∧ IsTall(q)}
  MaxAShort(p)        =  if ShortChildren(p) = ∅ then −1 else max {q.α | q ∈ ShortChildren(p)}
  MinATall(p)         =  if TallChildren(p) = ∅ then 2k + 1 else min {q.α | q ∈ TallChildren(p)}
  MinIDMinATall(p)    =  if TallChildren(p) = ∅ then p else min {q ∈ TallChildren(p) | q.α = MinATall(p)}
  Alpha(p)            =  if MaxAShort(p) + MinATall(p) ≤ 2k − 2 then MinATall(p) + 1 else MaxAShort(p) + 1
  Parent_CLR(p)       =  if IsClusterHead(p) then p else if p.α < k then Parent(p) else MinIDMinATall(p)
  Head_CLR(p)         =  if IsClusterHead(p) then p else p.parent_CLR.head_CLR
Actions:
  SetAlpha   ::  p.α ≠ Alpha(p)                  ⟶  p.α ← Alpha(p)
  SetParent  ::  p.parent_CLR ≠ Parent_CLR(p)    ⟶  p.parent_CLR ← Parent_CLR(p)
  SetHead    ::  p.head_CLR ≠ Head_CLR(p)        ⟶  p.head_CLR ← Head_CLR(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 *MaxAShort*(p) + *MinATall*$(p) > 2k - 2$, $p.\alpha =$ *MaxAShort*$(p) + 1$.
  In particular, If p is a leaf, then *MaxAShort*(p) + *MinATall*$(p) = 2k > 2k - 2$ and consequently, $p.\alpha = -1 + 1 = 0$.
- If *MaxAShort*(p) + *MinATall*$(p) \leq 2k - 2$, $p.\alpha =$ *MinATall*$(p) + 1$.

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 given 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.*

Constructing the 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.

B. 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

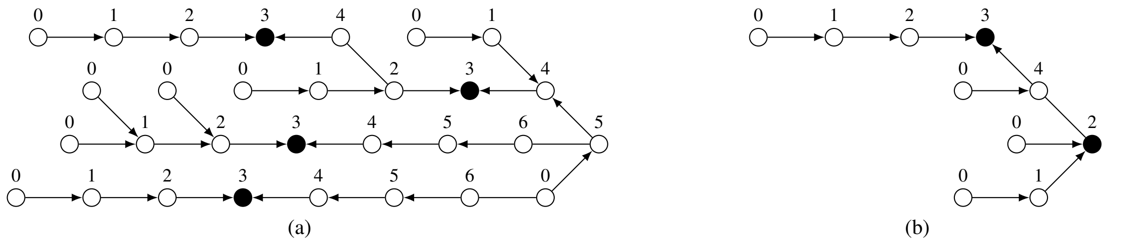

Figure 3: Examples of 3-Clustering using $\mathcal{CLR}(3)$. The root of each tree network is on the right, values of α are indicated, clusterheads are colored in black, and arrows represent local spanning tree of each k-cluster.

of $\mathcal{CLR}(k)$ is $O(H)$ rounds and the following lemma holds.

Lemma 3 *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 (Remark 1, Lemmas 4, and 5).

Remark 1 *Property 3.(a) follows immediately from the definition of α.*

Below, we prove Property 3.(b).

Lemma 4 *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. $\square$

We now prove Property 3.(c).

Lemma 5 *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 4, we are done.

Consider now any process p such that $p.\alpha \leq k$. We prove the lemma by strong backward 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 $q'.\alpha = \mathit{MaxAShort}(q') + 1$. As $q.\alpha < k$, q is *short* and $q.\alpha \leq \mathit{MaxAShort}(q')$. So:

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

  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 4, 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.

$\square$

We now use Property 3 to complete the correctness proof of $\mathcal{CLR}(k)$.

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 6 *In any terminal configuration of $\mathcal{CLR}(k)$, for every process p, there is a path $P = (p_1 = p, \ldots, p_m)$ such that: (1) $m \leq k$, (2) $\forall i \in [1..m-1], p_i.\mathrm{parent}_{CLR} = p_{i+1}$, (3) $p_m.\mathrm{parent}_{CLR} = p_m$, (4) $\forall i \in [1..m], p_i.\mathrm{head}_{CLR} = p_m$, (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 $IsClusterHead(p) = true$. Thus, by definition, $p.parent_{CLR} = p$ and $p.head_{CLR} = p$. 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 $Alpha(p)$, $p.\alpha = MinATall(p) + 1$, *i.e.*, there is some neighbor q of p such that $q.\alpha = MinATall(p)$. 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 $Q = (p_1 = q, \ldots, p_m)$ leading to a clusterhead p_m such that:

- $m \leq k$,
- $\forall i \in [1..m-1], p_i.parent_{CLR} = p_i + 1$,
- $p_m.parent_{CLR} = p_m$,
- $\forall i \in [1..m], p_i.head_{CLR} = p_m$.

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 $IsClusterHead(p) = true$ and the lemma holds. Consider now the case $p \neq r$ and note $q = \mathtt{Parent}(p)$. 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 $Alpha(q)$, $q.\alpha = MaxAShort(q) + 1$. 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 $Alpha(q)$, $q.\alpha = MinATall(q) + 1$ and:

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

Since $(p.\alpha \leq MaxAShort(q))$, then:

$$\begin{array}{ll} (p.\alpha + 1) + (q.\alpha - k) & \leq k \\ q.\alpha - k & \leq k - p.\alpha - 1 \\ |q.\alpha - k| & < |k - p.\alpha| \\ |q.\alpha - k| & < |p.\alpha - k| \end{array}$$

□

Lemma 7 *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 $IsClusterHead(p)$, $Parent_{CLR}(p)$, and $Head_{CLR}(p)$. 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 $Parent_{CLR}(p_j)$, for every $j \in [0..k-1]$, $p_j.parent_{CLR} = p_{j+1}$. By Definition of Macro $Head_{CLR}(p_j)$, for every $j \in [0..k-1]$, $p_j.head_{CLR} = p_{j+1}.head_{CLR} = p_k = p$. □

Lemma 8 *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 7, 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 6 and 8, we have:

Lemma 9 *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 3 and 9, 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 5), H is bounded 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.*

V. 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 10, 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 11, 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 10 *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 [12]. 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. $\square$

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 11 *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 7 (page 8), 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 3), 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:

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

$\square$

By Lemmas 10 and 11, 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:

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 [15].

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 10, 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 $\left(\frac{2}{\sqrt{3}}(\pi\lambda^2 k^2) + \frac{1}{2}(2\pi\lambda k) + 1\right)$ processes can be independent in this disk, and thus no more than 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 $\left(\frac{2\pi\lambda^2 k^2}{\sqrt{3}} + \pi\lambda k + 1\right)$ times the one of an optimum k-clustering Opt. By Lemma 11 and since $\frac{4\pi}{\sqrt{3}} \approx 7.2552$, we are done. $\square$

VI. 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 5) and, once the MIS Tree is built, the k-clustering is computed in $O(\mathcal{D})$ rounds by Theorem 3 (page 8) and Property 2 (page 5). 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 III. 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 $p_1, \ldots, p_n$ 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 $i > 1$ 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. It 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 [16].

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

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

VII. 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.

[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.

[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.

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

[5] S.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] A. K. Datta, S. Devismes, K. Heurtefeux, L. L. Larmore, and Y. Rivierre. Self-stabilizing small k-dominating sets. In *ICNC 2011*, pages 30–39. IEEE Computer Society, 2011.

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

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

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

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

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

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

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

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

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

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

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

[18] 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.

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