Theoretical Computer Science 626 (2016) 110–133

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Competitive self-stabilizing k-clustering ☆

Ajoy K. Datta [a], Stéphane Devismes [b,*], Karel Heurtefeux [b], Lawrence L. Larmore [a], Yvan Rivierre [b]

[a] School of Computer Science, University of Nevada, Las Vegas, USA
[b] VERIMAG UMR 5104, Université Joseph Fourier, Grenoble, France

ARTICLE INFO

Article history:
Received 27 October 2014
Received in revised form 19 November 2015
Accepted 12 February 2016
Available online 23 February 2016
Communicated by D. Peleg

Keywords:
Self-stabilization
k-Clustering
Competitiveness
Maximal independent set
MIS tree
$\mathcal{P}$-Completeness

ABSTRACT

In this paper, we give a silent self-stabilizing algorithm for constructing a k-clustering of any asynchronous connected network with unique IDs. Our algorithm stabilizes in $O(n)$ rounds, using $O(\log k + \log n)$ 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 Quasi-Unit Disk Graph (QUDG) with approximation ratio λ, then our algorithm is $7.2552\lambda^2 k + O(\lambda)$-*competitive*. In case of tree networks, our algorithm computes a k-clustering with the minimum number of clusters. 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 construction we use (called LFMIS) 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.

1. Introduction

Consider a simple undirected connected 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.

☆ A preliminary version of this work appeared in [1].

* Corresponding author.
E-mail address: Stephane.Devismes@imag.fr (S. Devismes).

http://dx.doi.org/10.1016/j.tcs.2016.02.010
0304-3975/

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 [2]. Instead, we give here a silent self-stabilizing distributed algorithm to construct $O(\frac{n}{k})$ k-clusters in an arbitrary asynchronous 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* [3] is a versatile property, enabling an algorithm to withstand transient faults in a distributed system. Indeed, 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 self-stabilizing distributed algorithms for finding a k-clustering of an asynchronous network, *e.g.*, [4–6]. The solution in [5] stabilizes in $O(k)$ rounds using $O(k \log n)$ space per process. The algorithm given in [7] stabilizes in $O(k.n)$ rounds using $O(k \log n)$ space per process. The algorithm given in [6] stabilizes in $O(n)$ rounds using $O(\log k + \log n)$ space per process. Note that, by definition, the set C of clusterheads for any k-clustering is a k-*dominating* set, that is, if every vertex of G is within k hops of some member of C. The k-dominating set computed by the algorithm given in [6] is also *minimal*, that is, none of its proper subsets is k-dominating. In the same paper, it is shown that every minimal k-dominating set contains at most $\max(1, n/\lceil \frac{k+1}{2} \rceil)$ nodes. In [8], an asynchronous silent self-stabilizing algorithm that computes a minimal k-dominating set of at most $\lceil \frac{n}{k+1} \rceil$ processes is given, this latter algorithm uses the one in [6] as module. 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 [8] stabilizes in $O(n)$ rounds using $O(\log k + \log n + k \log \frac{N}{k})$ bits per process, where N is any upper bound on n.

Note that all these aforementioned algorithms (*i.e.*, [5–8]) are written in the shared memory model and none of them is *competitive*. To the best of our knowledge, until now there has been no self-stabilizing competitive solution to the k-clustering problem.

There are several *non-self-stabilizing* distributed solutions for finding a k-clustering of a network [9–12]. Of those, only [10] deals with competitiveness. Moreover, they are all written in message-passing model. Deterministic solutions given in [9,10] 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 [9] are $O(k)$ and $O(k \log n)$, respectively. Fernandess and Malkhi [10] 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] Spohn and Garcia-Luna-Aceves [11] 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 [12] 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.

Detailed contribution and roadmap In the present paper, we give a silent self-stabilizing distributed algorithm for the k-clustering problem. This algorithm is written in the locally shared memory model. When the network is connected, asynchronous, and has unique node IDs, our algorithm:

- stabilizes in $O(n)$ rounds,
- requires $O(\log k + \log n)$ space per process, and
- constructs at most $\lceil \frac{n}{k+1} \rceil$ k-clusters.

To simplify the design of our solution, we write it as a *hierarchical collateral composition* of two sub-algorithms. That composition technique is defined in Section 2.

- Our first algorithm, proposed in Section 3, is silent and self-stabilizing, and constructs a particular kind of spanning tree, called an MIS *tree*. An MIS tree is a spanning tree whose processes at even levels form an MIS (maximal independent set) of the network. Our MIS tree algorithm is a straightforward self-stabilizing version of the non-self-stabilizing algorithm proposed by Alzoubi et al. [13].
- Our second algorithm, given in Section 4, is silent and self-stabilizing, and gives a k-clustering construction which works in *any* tree topology.

We then show that in several classes of network topologies, the number of k-clusters built by our algorithm can be more precisely analyzed:

- we prove in Section 4 that in tree networks, the computed k-clustering is minimum, *i.e.*, has the minimum possible number of k-clusters,

[1] Actually, in [10], 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 a k-cluster.

- in Section 5, we analyze the competitiveness of our k-clustering algorithm in UDGs and QUDGs because these topologies are commonly used to model wireless sensor networks:
  - In a UDG, our algorithm is $7.2552k + O(1)$-*competitive*,
  - in an QUDG with approximation ratio λ, our algorithm is $7.2552\lambda^2 k + O(\lambda)$-competitive.

We then partially answer the question: "Is it possible to reduce the time complexity of our algorithm to $O(\mathcal{D})$ rounds (the trivial lower bound) where $\mathcal{D}$ is the diameter of the network, while retaining its competitiveness for the UDG and QUDG cases?"

We show in Section 5 that our competitiveness result for both UDGs and QUDGs depends on the fact that we use the Alzoubi et al. MIS tree. The construction of that tree is the time bottleneck of our k-clustering algorithm, since that construction takes $\Theta(n)$ rounds in the worst case. The remainder of our algorithm takes $O(\mathcal{D})$ rounds. More precisely:

- our algorithm constructs, in $\Theta(n)$ rounds, an MIS spanning tree of height at most $2\mathcal{D}$, as shown in Section 3;
- our algorithm constructs, in $O(H)$ rounds, a k-clustering on any tree, where H the height of the (spanning) tree in which it is deployed, as shown in Section 4.

In Section 6, we show that the time complexity of our self-stabilizing MIS tree algorithm could be hard to enhance since whether a given process is part of the Alzoubi et al. MIS tree is a $\mathcal{P}$-*complete* problem.

Finally, in Section 7, we give some concluding remarks and perspectives.

2. Preliminaries

Computational model We consider networks made of n processes. Each process can directly communicate with a subset of other processes, called *neighbors*. We denote by $\mathcal{N}_p$ the set of neighbors of process p. Communications are assumed to be *bidirectional*, that is, for all processes p, q we have: $q \in \mathcal{N}_p \Leftrightarrow p \in \mathcal{N}_q$. Hence, as commonly done in the literature, we model a distributed system as a simple undirected connected graph $G = (V, E)$, where V is the set of processes and E is a set of edges representing (direct) communication relations.

Processes have unique IDs. By abuse of notation, we shall identify any process with its ID, whenever convenient. If b bits are used to store each identifier, then the space complexity of our algorithm will be $\Omega(b)$ per process, but henceforth, as is commonly done in the literature, we will assume that $b = O(\log n)$.

We assume the *shared memory model* of computation [3], where a process communicates with its neighbors using locally shared variables (henceforth called variables). Each process can read its own variables and those of its neighbors, but can write only to its own variables. Each process operates according to its (local) *program*. We call *(distributed) algorithm* $\mathcal{A}$ a collection of n *programs*, each one operating on a single process. In the following, we will denote the local program of Process p in the distributed algorithm $\mathcal{A}$ by $\mathcal{A}(p)$. The *program* of each process is a set of actions:

$$\langle label \rangle :: \langle guard \rangle \rightarrow \langle statement \rangle$$

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 $\mathcal{A}$.

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 $\mathcal{A}$. Each step $\gamma \mapsto \gamma'$ 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* [14].

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.

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* [15]. This definition captures the execution rate of the slowest process in every execution. The first *round* of an execution e, noted e', is the minimal prefix of e in which every process that is enabled in the initial configuration either executes an action or becomes neutralized. Let e'' be the suffix of e starting from the last configuration of e'. The second round of e is the first round of e'', and so forth.

Self-stabilization and silence Let $\mathcal{A}$ be a distributed algorithm and P be a predicate over the configurations of $\mathcal{A}$. $\mathcal{A}$ is *self-stabilizing w.r.t.* P if there exists a non-empty subset $\mathcal{S}$ of configurations of $\mathcal{A}$ such that:

- $\forall \gamma \in \mathcal{S}$, $P(\gamma)$. *(Correction)*
- For each possible step $\gamma \mapsto \gamma'$ of $\mathcal{A}$, $\gamma \in \mathcal{S} \Rightarrow \gamma' \in \mathcal{S}$. *(Closure)*
- Each execution of $\mathcal{A}$ (starting from an arbitrary configuration) contains a configuration of $\mathcal{S}$. *(Convergence)*

The configurations of $\mathcal{S}$ are said to be *legitimate*, and other configurations are called *illegitimate*.

We say that an algorithm is *silent* [16] 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. In this paper, we are interested in silent self-stabilizing algorithms. To show that an algorithm $\mathcal{A}$ is silent, and self-stabilizing *w.r.t.* P, it is sufficient to show that (1) every execution of $\mathcal{A}$ is finite and (2) every terminal configuration of $\mathcal{A}$ satisfies P.

Composition To simplify the design of our algorithm, we use *hierarchical collateral composition* [8] which is a variant of *collateral composition* [17]. 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 $\mathcal{B}(p)$ (for every process p) so that p executes an action of $\mathcal{B}(p)$ only when it has no enabled action in $\mathcal{A}(p)$.

Definition 1 *(Hierarchical collateral composition).* Let $\mathcal{A}$ and $\mathcal{B}$ be two (distributed) algorithms such that no variable written by $\mathcal{B}$ appears in $\mathcal{A}$. In the *hierarchical collateral composition* of $\mathcal{A}$ and $\mathcal{B}$, noted $\mathcal{B} \circ \mathcal{A}$, the (local) program of every process p, $\mathcal{B}(p) \circ \mathcal{A}(p)$, is defined as follows:

- $\mathcal{B}(p) \circ \mathcal{A}(p)$ contains all variables of $\mathcal{A}(p)$ and $\mathcal{B}(p)$.
- $\mathcal{B}(p) \circ \mathcal{A}(p)$ contains all actions of $\mathcal{A}(p)$.
- For every action $G_i \rightarrow S_i$ of $\mathcal{B}(p)$, $\mathcal{B}(p) \circ \mathcal{A}(p)$ contains the action $\neg C_p \wedge G_i \rightarrow S_i$ where C_p is the disjunction of all guards of actions in $\mathcal{A}(p)$.

We recall a theorem from [8] 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. *P assuming a weakly fair daemon if the following conditions hold:*

- *$\mathcal{A}$ is a silent (self-stabilizing) algorithm under a weakly fair daemon.*
- *$\mathcal{B}$ is self-stabilizing* w.r.t. *P assuming a weakly fair daemon starting from any configuration where no action of $\mathcal{A}$ is enabled ever.*[2]

Nick's class $\mathcal{NC}$ (stand for *Nick's class*) [18] 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 $\mathcal{NC}$.

Recall that $\mathcal{P}$ is the set of all problems that can be deterministically solved in polynomial time. $\mathcal{NC} \subseteq \mathcal{P}$ because a polylogarithmic time parallel computation with polynomially many processors can be emulated by polynomial-time sequential computation. The question, "Is $\mathcal{NC} = \mathcal{P}$?" is still open and considered to be in the same class of difficulty as the question of whether $\mathcal{P} = \mathcal{NP}$. Most researchers suspect that $\mathcal{NC} \neq \mathcal{P}$, meaning believe there to be tractable problems which are "inherently sequential," and cannot be executed in polylogarithmic time up by using parallelism.

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 *$\mathcal{NC}$-reduction* of $\mathbb{B}$ to $\mathbb{A}$, *i.e.*, a reduction that 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 $\mathcal{NC}$.

Now, if we make the usual assumption that $\mathcal{NC} \neq \mathcal{P}$, then any $\mathcal{P}$-complete problem belongs to $\mathcal{P} \setminus \mathcal{NC}$, meaning that the problem is "inherently sequential." Hence, just as we can justify giving up the search for a polynomial time algorithm for any problem that we can prove to be $\mathcal{NP}$-complete, we can justify giving up the search for a fast parallel algorithm for a problem if we can prove that it is $\mathcal{P}$-complete.

3. The MIS tree

In this section, we first recall the definition of MIS tree (for Maximal Independent Set tree), introduced in [13]. Then, we give a silent self-stabilizing algorithm that computes an MIS tree in any arbitrary identified network within $O(n)$ rounds, this algorithm is a straightforward self-stabilizing version of the non-self-stabilizing algorithm of Alzoubi et al. [13]. 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], *i.e*, it is the *lexically first MIS tree*.

[2] Recall that in such a configuration, the specification of $\mathcal{A}$ is satisfied.

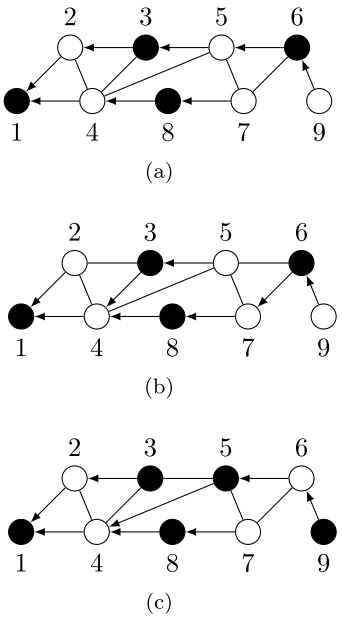

Fig. 1. Examples. Process 1 is the root. Arrows represent the parent pointers for each non-root process. Processes at even levels are in black. The MIS tree given in (a) is a LFMIST. The MIS tree given in (b) is not lexically first. The tree given in (c) is not an MIS tree.

3.1. Definition of MIS tree

The subset $I \subseteq V$ is an independent set of $G = (V, E)$ 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 in T. 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 $\texttt{Level}(p) = \texttt{Level}(q) + 1$.

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.

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 encoding is first in the lexical order of the encodings of all MIS trees of G with root r. For example, two MIS trees are given in Figs. 1a and 1b: their respective sets of processes at even level (black nodes) form maximal independent sets. However, only the tree given in Fig. 1a is a LFMIST. Its encoding is $(1, 1, 2, 1, 3, 5, 8, 4, 6)$, while the encoding of MIS the tree given in Fig. 1b is $(1, 1, 4, 1, 3, 7, 8, 4, 6)$, and $(1, 1, 4, 1, 3, 7, 8, 4, 6) > (1, 1, 2, 1, 3, 5, 8, 4, 6)$ in the lexical order. The tree given in Fig. 1c is not an MIS tree, indeed the set of processes at even level is not a maximal independent set, as 3 and 5 are neighbors.

3.2. The algorithm to construct an MIS tree

We now give a silent self-stabilizing algorithm to construct an MIS tree (actually a LFMIST) in $O(n)$ rounds. It is defined as the hierarchical collateral composition $\mathcal{MIST} \circ \mathcal{BFST}$, where $\mathcal{BFST}$ is a silent self-stabilizing algorithm that constructs a breadth-first spanning tree (BFS tree), and $\mathcal{MIST}$ is an algorithm that uses the BFS tree to compute an MIS tree of the network.

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 $\|p, r\|$, *i.e.*, the distance from p to r in G.

Let $\mathcal{BFST}$ be a 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. One of the first silent self-stabilizing algorithm for that problem is given in [19]. However, it was designed for arbitrary rooted networks. The silent self-stabilizing algorithm for identified networks given in [20] can be used to implement $\mathcal{BFST}$. Actually, this algorithm is a leader election, but, as do most of the existing silent self-stabilizing leader election algorithms, it also builds a BFS tree that is rooted at the elected node. This algorithm stabilizes in $O(n)$ rounds using $O(\log n)$ bits per process, and does not require processes to know any upper bound on the size n or the diameter $\mathcal{D}$ of the network.

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 pairs. Using the outputs of $\mathcal{BFST}$, $\mathcal{MIST}$ computes the MIS tree of the network which 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 $p.par$, 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 its status, dominator or dominated, each process uses a *key*, noted $Key(p)$, 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 keys and the status of its neighbors, p decides its status as follows: p is a dominator if and only if each neighbor q is either dominated or satisfies $Key(q) > Key(p)$, where $>$ is the strict lexical ordering. According to this rule, the root of the BFS tree is the node of minimum key and consequently is eventually definitely a dominator. All its neighbors becomes dominated, and so on. Hence, eventually, the set of dominator processes is a maximal independent set.

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, and of minimum key. 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.

Input : $\texttt{Level}_{\texttt{BFS}}(p) \in \mathbb{N}$
Variables: $p.dominator$: Boolean ; $p.par \in \mathcal{N}_p \cup \{p\}$
Macros:

$Key(p)$	=	$(\texttt{Level}_{\texttt{BFS}}(p), p)$
$Dominator(p)$	=	$\forall q \in \mathcal{N}_p, \neg q.dominator \vee\ Key(q) > Key(p)$
$Par(p)$	=	**if** $\texttt{Level}_{\texttt{BFS}}(p) = 0$ **then** p
		else $q \in \mathcal{N}_p \mid Key(q) = \min\{Key(q') \mid q' \in \mathcal{N}_p \wedge\ q'.dominator \neq p.dominator\}$

Actions:

SetDominator	::	$p.dominator \neq Dominator(p)$	$\rightarrow$	$p.dominator \leftarrow Dominator(p)$
SetParMIS	::	$p.dominator = Dominator(p) \wedge\ p.par \neq Par(p)$	$\rightarrow$	$p.par \leftarrow Par(p)$

Correctness and complexity analysis According to Theorem 1, to show the correctness of $\mathcal{MIST} \circ \mathcal{BFST}$, we show that $\mathcal{MIST}$ constructs 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.

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, $Key(p)$ is fixed forever for every process p. Let $p_1, \ldots, p_n$ the list of processes ordered by $\prec$ (the lexical ordering *w.r.t.* keys) in γ. We show the lemma by induction on the rank of every process in the ordering.

- **Base case:** In γ, $p_1 = r$ and $Key(p_1) = (0, r)$. So, 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 key that is fixed forever; moreover if $Key(q) < Key(p_{j+1})$, then the value $q.dominator$ is fixed forever by the 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 executed, 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 $Key(p)$ and $p.dominator$ are fixed forever, for every process p. Now, for all processes, the guard of action SetParMIS only depends on these values. So, after at most one additional round, no action of $\mathcal{MIST}$ can ever again be enabled, and we are done. □

We now consider any terminal configuration γ of $\mathcal{MIST} \circ \mathcal{BFST}$. Let I be 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 having their respective *dominator* variable equal to *true*. Then, either $Key(p) < Key(q)$ or $Key(q) < Key(p)$. In the first case, Action SetDominator is enabled at q, in the latter Action SetDominator is enabled at p, a 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, a contradiction. □

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

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

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

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 $q = \texttt{Par}_{\texttt{BFS}}(p)$. Note that $\texttt{Level}_{\texttt{BFS}}(\texttt{Par}_{\texttt{BFS}}(p)) = \texttt{Level}_{\texttt{BFS}}(p) - 1$. Thus, by definition of the macro $Par(p)$, $\texttt{Level}_{\texttt{BFS}}(p.par) = \texttt{Level}_{\texttt{BFS}}(\texttt{Par}_{\texttt{BFS}}(p))$. Consequently, $Key(p.par) < Key(p)$.
- $p \notin I$. Then $\neg Dominator(p)$. Now, as no two processes have equal key, we have $\exists q \in \mathcal{N}_p$, $Key(p) > Key(q) \wedge q.dominator$. So, $Key(p.par) \leq Key(q)$ by definition of Macro $Par(p)$. Consequently, $Key(p.par) < Key(p)$. □

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 $\{\{p, p.par\} \mid p \in V \setminus \{r\}\}$ defined in γ. (Recall that r is the unique process such that $r.par = r$ 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 an elementary cycle in $C = (c_0, c_1, \ldots, c_m = c_0)$ such that $\forall i \in [0..m-1]$, $c_i.par = c_{i+1}$ and $m > 0$. By Remark 1, $r \notin C$. By Lemma 4, $\forall i \in [0..m-1]$, $Key(c_i) < Key(c_{i+1})$ (since $c_i.par = c_{i+1}$). By transitivity, $Key(c_0) < Key(c_m)$, that is, $Key(c_0) < Key(c_0)$, a 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 $Par(p)$. 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. □

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 $q.dominator = true$ for all q such that $\texttt{Level}_{\texttt{MIS}}(q) = L - 1$.

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

We can require that $\mathcal{BFST}$ stabilize in $O(n)$ rounds and use $O(\log n)$ space per process [20]. So, 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. To show the property, we need the following technical lemma.

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

Proof. As p is a dominator process, none of its neighbors is a dominator, by Lemma 3. Since p is not the root, $\texttt{Par}_{\texttt{BFS}}(p)$ is defined. To sum up, $\texttt{Par}_{\texttt{BFS}}(p) \in \mathcal{N}_p$ and $\texttt{Level}_{\texttt{BFS}}(\texttt{Par}_{\texttt{BFS}}(p)) = \texttt{Level}_{\texttt{BFS}}(p) - 1$, 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 $Key(q) < Key(p)$. By Macro $Par(p)$, we are done. □

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.

Proof. Let H be the height of T_{BFS}. 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 $p_j = p_{j+1}.par$ 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 $p_j.dominator = true$ if and only if j is even, for all $j < \ell$.

Since $Key(p_{j+1}) > Key(p_j)$ (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, $\texttt{Level}_{\texttt{BFS}}(p.par) < \texttt{Level}_{\texttt{BFS}}(p)$ for any dominator process $p \neq r$. Thus:

(b) $\texttt{Level}_{\texttt{BFS}}(p_{j+1}) > \texttt{Level}_{\texttt{BFS}}(p_j)$ for all odd 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}.

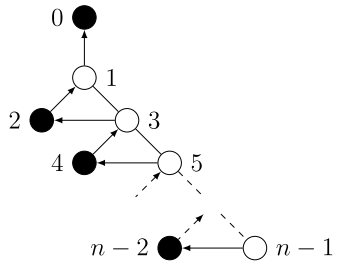

Fig. 2. Worst case example for MIS tree height.

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, $\ell \leq 2 \times H \leq 2 \times \mathcal{D}$. □

Fig. 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 the level of the parent in T_{MIS} of each odd process p is the level of p 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 $\lceil \frac{n}{k+1} \rceil$ k-clusters

In this section, we present a silent self-stabilizing algorithm, called $\mathcal{CLR}(k)$, which constructs a k-clustering in any directed tree T. Its stabilization time is $O(H)$ rounds, where H is the height of T. At the end of this section we show that this clustering is optimal (*i.e.*, minimum in terms of number of clusters) in T. 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 $\lceil \frac{n}{k+1} \rceil$ distinct k-clusters in any arbitrary network. Moreover, we will see in Section 5 that the composition between $\mathcal{CLR}(k)$ and our spanning tree construction $\mathcal{MIST} \circ \mathcal{BFST}$ is competitive in both UDG and QUDG networks. The stabilization time of $\mathcal{CLR}(k) \circ \mathcal{MIST} \circ \mathcal{BFST}$ is $O(n)$ rounds and its memory requirement is $O(\log k + \log n)$ space per process. We conclude the section with few experimental results.

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

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 $\lceil \frac{n}{k+1} \rceil$. 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 $\lceil \frac{n}{k+1} \rceil$ k-clusters. $\mathcal{CLR}(k)$ uses the following three variables in the code of each process p:

- $p.\alpha$, *an integer in the range* $[0..2k]$. Once correctly computed, the value of $p.\alpha$ is equal to $\|p, q\|$, where q is the furthest process in $T(p)$ (the subtree rooted at p) which is in the same k-cluster as p. Consequently, 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 $p = r$, see Predicate *IsClusterHead*(p). Further details are given in the next paragraph.
- $p.par_{CLR} \in \mathcal{N}_p \cup \{p\}$. In any terminal configuration, $p.par_{CLR}$ is the parent of p in its k-cluster, unless p is a clusterhead, in which case $p.par_{CLR} = p$. These variables are used to define a local BFS structure for each cluster, rooted at its clusterhead.
- $p.hd_{CLR} \in V$. In any terminal configuration, $p.hd_{CLR}$ 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, in a bottom-up fashion starting from the leaves of T. As previously explained, *Dom* is defined using the values of $p.\alpha$ for all p.

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

Input: $\texttt{Par}(p) \in \mathcal{N}_p \cup \{p\}$
Variables: $p.\alpha \in [0..2k]$; $p.par_{CLR} \in \mathcal{N}_p \cup \{p\}$; $p.hd_{CLR} \in V$
Macros:

$IsShort(p)$	$\equiv$	$p.\alpha < k$
$IsTall(p)$	$\equiv$	$p.\alpha \geq k$
$IsClusterHead(p)$	$\equiv$	$(p.\alpha = k) \vee (IsShort(p) \wedge (p = r))$
$ShortChildren(p)$	$=$	$\{q \in \mathcal{N}_p \mid (\texttt{Par}(q) = p) \wedge IsShort(q)\}$
$TallChildren(p)$	$=$	$\{q \in \mathcal{N}_p \mid (\texttt{Par}(q) = p) \wedge IsTall(q)\}$
$MaxAShort(p)$	$=$	**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** $MaxAShort(p)+1$
$Par_{CLR}(p)$	$=$	**if** $IsClusterHead(p)$ **then** p **else if** $p.\alpha < k$ **then** $\texttt{Par}(p)$ **else** $MinIDMinATall(p)$
$Hd_{CLR}(p)$	$=$	**if** $IsClusterHead(p)$ **then** p **else** $p.par_{CLR}.hd_{CLR}$

Actions:

SetAlpha	::	$p.\alpha \neq Alpha(p)$	$\rightarrow$	$p.\alpha \leftarrow Alpha(p)$
SetParCLR	::	$p.\alpha = Alpha(p) \wedge p.par_{CLR} \neq Par_{CLR}(p)$	$\rightarrow$	$p.par_{CLR} \leftarrow Par_{CLR}(p)$
SetHead	::	$p.\alpha = Alpha(p) \wedge p.par_{CLR} = Par_{CLR}(p) \wedge p.hd_{CLR} \neq Hd_{CLR}(p)$	$\rightarrow$	$p.hd_{CLR} \leftarrow Hd_{CLR}(p)$

We now give more details about the value and the computation of $p.\alpha$ for all p. Consider any terminal configuration. We recall that in this configuration, $p.\alpha = \|p,q\|$, where q is the furthest process in $T(p)$ that is in the same k-cluster as p.

We divide processes into *short* and *tall* according to the value of their α-variable:

(i) If p satisfies $IsShort(p)$, *i.e.*, $p.\alpha < k$, 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 $k - p.\alpha$. See, for example, in Configuration (VI) of Fig. 4, $k = 2$ and $g.\alpha = 0$ mean that the clusterhead of g is at most at distance $k - 0 = 2$, now its clusterhead d is at distance 1.
In the latter case ($p = r$), p may not be k-dominated by any process of Dom inside its subtree and, by definition, there is no process outside its subtree, indeed $T(p) = T$, see the root in Configuration (VI) of Fig. 4. Thus, p must be placed in Dom.

(ii) If p satisfies $IsTall(p)$, *i.e.*, $p.\alpha \geq k$, 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. See, for example, in Configuration (VI) of Fig. 4, $k = 2$ and $c.\alpha = 3$ mean that the clusterhead of c, here d, is $3 - k = 1$ hop below c.
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 macro $Alpha(p)$. This latter is based on 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* child, $MaxAShort(p)$ returns -1.
- $MinATall(p)$ returns the minimum value of $q.\alpha$ for all *tall* children q of p. If p has no *tall* child, $MinATall(p)$ returns $2k+1$.

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

- If $MaxAShort(p) + MinATall(p) > 2k-2$, $p.\alpha = MaxAShort(p) + 1$.
- If $MaxAShort(p) + MinATall(p) \leq 2k-2$, $p.\alpha = MinATall(p) + 1$.

Consider a leaf f. Since f has no children, $MaxAShort(f) + MinATall(f) = -1 + 2k + 1 > 2k - 2$. Thus, $f.\alpha = -1 + 1 = 0$, which corresponds to the distance between f and its furthest descendant that will be in its cluster (f itself).

Consider now an internal process p and assume that the α-variables of all its children are correctly evaluated. p should choose a clusterhead that will be either (1) in its subtree (in this case, p will be *tall*), or (2) outside its subtree (in this case p will be *short*). Since the computations are done bottom-up, we should preferably make the choice (1) to reduce the number of clusterheads.

Let q be a *short* child of p. From (i), the path from q to its clusterhead goes through p. Thus, to prevent cycle creation,

($*$) p must not choose a clusterhead which is in the subtree of any of its short children.

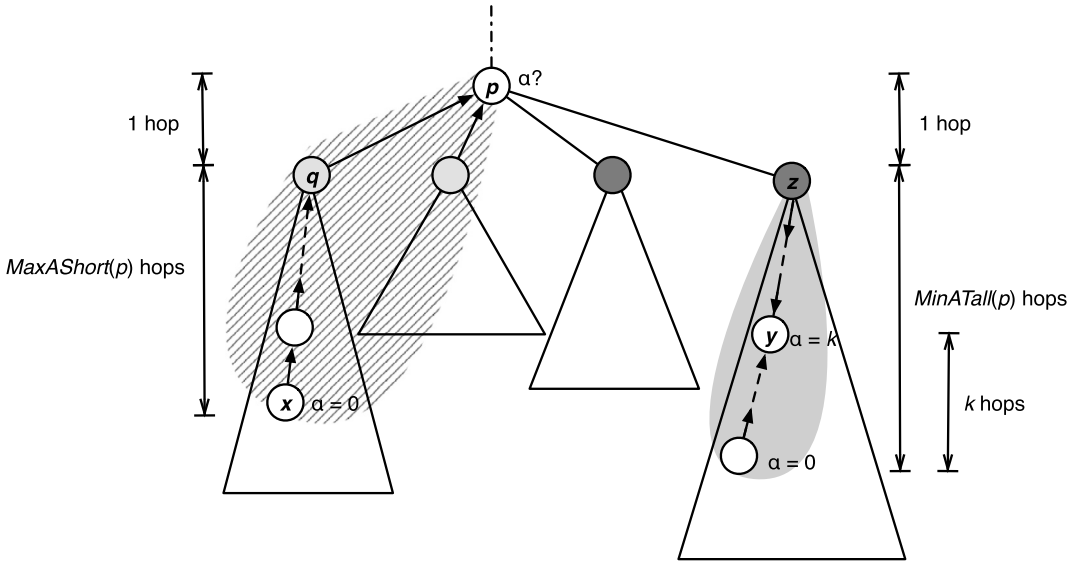

Fig. 3. Illustrative example: Light grey nodes are *short* children of p; grey nodes are *tall* children of p. The shaded area shows the nodes that already chose the same cluster as p. The light grey area shows the nodes that already choose the same cluster as z.

From now on, follow the illustrative example given in Fig. 3. Let x be the furthest process that is both in the subtree of some *short* child of p and in the same cluster as p. Let q be the *short* child of p such that $x \in T(q)$. Then, from (i), x is at distance $MaxAShort(p)+1$ from p. Two cases are then possible:

- $MaxAShort(p) + MinATall(p) > 2k-2$. If p chooses a node y of its subtree as clusterhead, then from $(*)$, the path from p to its clusterhead should go through one of its *tall* children. So, p will be at least at distance $MinATall(p)-k+1$ from that clusterhead, from (ii). Now, in this case, x will be at least at distance $MaxAShort(p)+1+MinATall(p)-k+1 > 2k-2-k+2=k$ from the clusterhead y, this violates the definition of k-clustering. Thus, p should necessarily choose its clusterhead outside the subtrees of any of its children (that is, either p declares itself as clusterhead or chooses an ancestor as clusterhead). From (i) and (ii), this means that all nodes in the subtrees of the *tall* children of p adopt a different cluster from p, and consequently the node x is then the furthest node that belongs to both $T(p)$ and the cluster of p. This implies that $p.\alpha = \|p,x\| = MaxAShort(p)+1$.
- $MaxAShort(p) + MinATall(p) \leq 2k-2$. Let z be a *tall* child of p such that $z.\alpha = MinATall(p)$. Unlike the previous case, p can choose a node y in the subtree of z as clusterhead. Indeed, in this case, x will be at distance $MaxAShort(p)+1+MinATall(p)-k+1 \leq 2k-2-k+2=k$ from y. Hence, the nodes (other than p) that are both in the subtree of p and in its cluster will be either nodes in subtrees of short children of p or nodes in $T(z)$. Since by definition, $MinATall(p) > MaxAShort(p)$, the furthest node that belongs to both $T(p)$ and the cluster of p will be at distance $MinATall(p)+1$ from p, *i.e.*, $p.\alpha = MinATall(p)+1$.

Using Fig. 4, we now detail an example of computation of α-values for $k=2$. In Fig. 4, the root of the tree network is the rightmost node, node a. Recall that the computation of α-values is bottom-up. In the following explanation, we only consider at each round the processes that are guaranteed to take their final α-value, some others may move but these moves have no impact on the reasoning. First, regardless the initial configuration, our algorithm ensures that every leaf has its final α-value at the end of the first round (Configuration I): every leaf $x \in \{b,g,i,k\}$ satisfies $MaxAShort(x)+MinATall(x) = -1+2k+1 = 4 > 2k-2 = 2$. Thus, $x.\alpha$ takes value $MaxAShort(x)+1 = -1+1 = 0$. Of course, the clusterhead of each leaf will be up in the tree. During the second round (Configuration II), nodes f and j get their final α-value, 1, as all the α-values of all their respective children are now fixed. Indeed, for example, $MaxAShort(f)+MinATall(f) = 0+2k+1 = 5 > 2$, so f satisfies $f.\alpha = MaxAShort(f)+1 = 0+1 = 1$. f and j being short, their respective clusterheads will be up in the tree. At the end of the third round (Configuration III), h and d have their final α-value, 2. For example, $MaxAShort(h)+MinATall(h) = 1+2k+1 = 6 > 2$, so, $h.\alpha$ takes value $MaxAShort(h)+1 = 1+1 = 2$. Notice that $h.\alpha = 2$ and $d.\alpha = 2$ means that both h and d are clusterheads. In particular, we already know that h (resp. d) is the clusterhead of k and j (resp. of i, f, and g). At the end of the fourth round (Configuration IV), only e is guaranteed to have its final α-value: $MaxAShort(e)+MinATall(e) = -1+2 \leq 2$. So, $e.\alpha$ takes value $MinATall(e)+1 = 2+1 = 3$. So, e is tall and its clusterhead is below in the tree: h. At the end of the fifth round (Configuration V), only c is guaranteed to have its final α-value: $MaxAShort(c)+MinATall(c) = -1+2 = 1 \leq 2$. So, $c.\alpha$ takes value $MinATall(c)+1 = 2+1 = 3$, which means that c is tall. The clusterhead of c is then a clusterhead already defined in the subtree of its smallest tall children, here d. Finally, a has its final value at the end of the sixth round (Configuration VI): $MaxAShort(a)+MinATall(a) = 0+3 = 3 > 2$. So $a.\alpha$ takes value $MaxAShort(a)+1 = 0+1 = 1$. As a is a short root, a is a

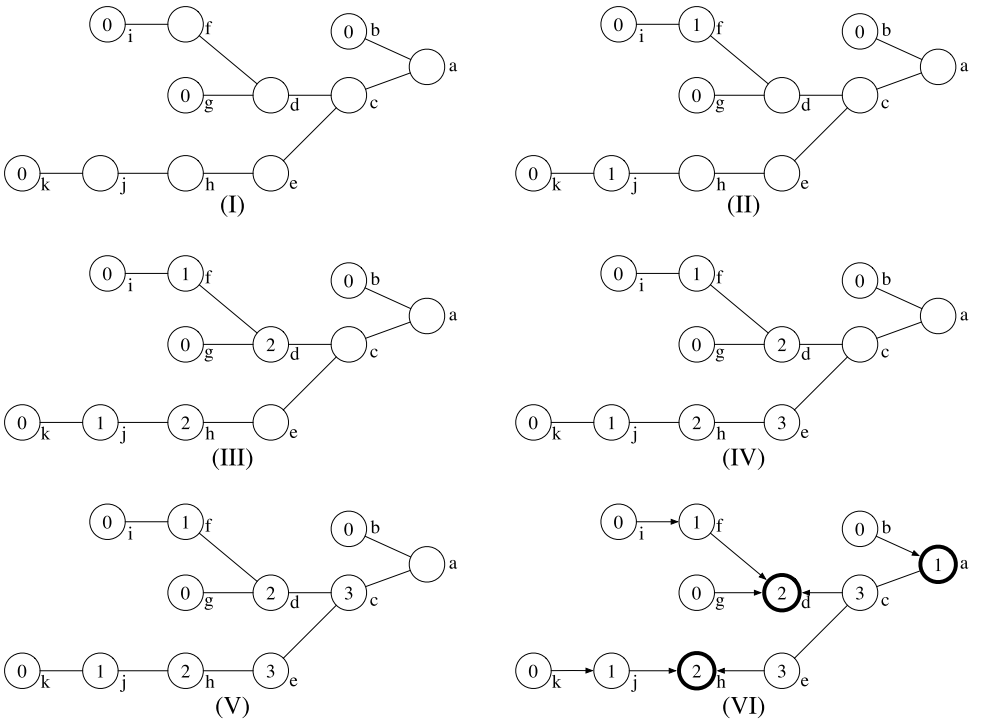

Fig. 4. Computation of α for $k = 2$. The root of the tree network is the rightmost node. α-values are given inside the nodes; when no value is given α is arbitrary. Configuration (VI) is terminal: bold circles represent clusterheads and arrows represent local spanning tree of each k-cluster.

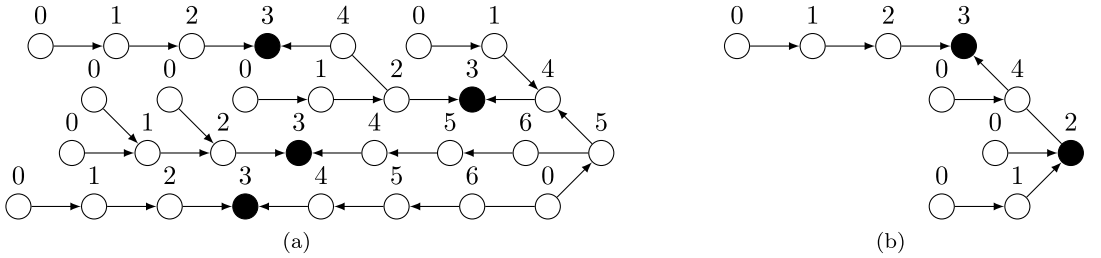

Fig. 5. Examples of k-Clustering using $\mathcal{CLR}(3)$, where $k = 3$. The root of each tree network is on the right, values of α are indicated, clusterheads are colored in black, and arrows represent local spanning tree of each k-cluster.

clusterhead. Hence, in Configuration (VI), three 2-clusters are defined $\{c, d, f, g, i\}$, $\{e, h, j, k\}$, and $\{a, b\}$ with d, h, and a as respective clusterheads.

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 Fig. 5, and will be proven in Subsection 4.2.

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 SetParCLR and SetHead. Other processes p designate their parent using Action SetParCLR 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 (we use IDs to break

ties, see *MinIDMinATall*(p)). Finally, identifiers of clusterheads are propagated in a top-down fashion in their k-cluster using Action SetHead, see macro $Hd_{CLR}(p)$.

Two examples of 3-clustering using $\mathcal{CLR}(3)$ are given in Fig. 5. In Fig. 5a, the root is a *tall* process, consequently it is not a clusterhead. In Fig. 5b, 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, where H is the height of T. After that, one additional round is necessary to fix the Par_{CLR} variables, because the values of these variables only depend on the α variables. Finally, the hd_{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 $p.par_{CLR}$ is fixed forever within $H+2$ rounds.*

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

Lemma 10. *In every configuration where all par_{CLR} and α variables are fixed forever, there is no directed cycle constituted of directed edges of the form $(p, p.par_{CLR})$ 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 $p.par_{CLR} = q$ and $q.par_{CLR} = p$. Without loss of generality, assume that q is a child of p. Then, by definition of Macro $Par_{CLR}(q)$, $q.\alpha < k$. By definition of Macro $Par_{CLR}(p)$, $q.\alpha \geq k$, a contradiction. □

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

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

Then, for every process p, the variable $p.hd_{CLR}$ only depends on $p.par_{CLR}.hd_{CLR}$ and some fixed variables.

For every process p such that $p.par_{CLR} = p$, $p.hd_{CLR}$ is fixed forever in at most one additional round. Then, changes on hd_{CLR} can be propagated from node p to its neighbor q only if $q.par_{CLR} = p$. 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 now consider any terminal configuration of $\mathcal{CLR}(k)$ and show that such a configuration is legitimate. The proof begins by formally establishing the three claims given in Property 3 (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 $k < p.\alpha < a$ and 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 the 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 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 $q'.\alpha = MaxAShort(q') + 1$. 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 the induction hypothesis, there is a member of Dom which is within $k - q'.\alpha$ hops of q'. This process is within $k - q'.\alpha + 1$ hops from q. Now:

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

  This process is within $|q.\alpha - k|$ hops from q and we are done.
- Otherwise, $q'.\alpha = MinATall(q') + 1$ 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:

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

  Hence:

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

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

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 15. *In any terminal configuration of $\mathcal{CLR}(k)$, for every process p, there is a path $P = (p_0 = p, \ldots, p_m)$ such that:*

(1) $m \leq |p.\alpha - k| \leq k$,
(2) $\forall i \in [0..m-1]$, $p_i.par_{CLR} = p_{i+1}$,
(3) $p_m.par_{CLR} = p_m$,
(4) $\forall i \in [0..m]$, $p_i.hd_{CLR} = p_m$,
(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 $|p.\alpha - k| \in [0..k]$ always.

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

Assume the lemma holds for every q such that $|q.\alpha - k| < a$, and consider a process p 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 $q.\alpha = MinATall(p)$, hence $p.\alpha = q.\alpha + 1$. Consider the process of smallest identifier. Since $p.\alpha - k = a$, it follows that $q.\alpha + 1 - k = a$, that is, $q.\alpha - k = a - 1 < a$. By the induction hypothesis, there is a path $Q = (p_0 = q, \ldots, p_m)$ leading to a clusterhead p_m such that:

- $m \leq |q.\alpha - k| \leq k$,
- $\forall i \in [0..m-1]$, $p_i.par_{CLR} = p_{i+1}$,
- $p_m.par_{CLR} = p_m$, and
- $\forall i \in [0..m]$, $p_i.hd_{CLR} = p_m$.

By definition of $Par_{CLR}(p)$ and $Head_{CLR}(p)$, $p.par_{CLR} = q$ and $p.hd_{CLR} = p_m$. Then, as $q.\alpha \geq k$, $|q.\alpha - k| + 1 = |q.\alpha - k + 1| = |p.\alpha - k|$. Hence, the path $p, p_0 = q, \ldots, p_m$ has length at most $|p.\alpha - k|$, and we are done.

Otherwise, $p.\alpha < k$. If $p = r$, then $IsClusterHead(p) = true$ and the lemma holds. Consider now the case $p \neq r$ and note $q = \texttt{Par}(p)$. By definition of $Par_{CLR}(p)$, $p.par_{CLR} = q$. By definition of $Head_{CLR}(p)$, $p.hdCLR = q.hd_{CLR}$. 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 $p.\alpha < q.\alpha \leq k$, $|q.\alpha - k| < |p.\alpha - k|$.
- $q.\alpha > k$, 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:

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

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, $p.\alpha = k$, $p.par_{CLR} = p$, and $p.hd_{CLR} = p$ by definition of $IsClusterHead(p)$, $Par_{CLR}(p)$, and $Hd_{CLR}(p)$. 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 $Par_{CLR}(p_j)$, for every $j \in [0..k-1]$, $p_j.par_{CLR} = p_{j+1}$. By Definition of Macro $Hd_{CLR}(p_j)$, for every $j \in [0..k-1]$, $p_j.hd_{CLR} = p_{j+1}.hd_{CLR} = p_k = p$. $\square$

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

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 $\lceil \frac{n}{k+1} \rceil$ 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 $\lceil \frac{n}{k+1} \rceil$ 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 117), 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 $\lceil \frac{n}{k+1} \rceil$ distinct k-clusters within $O(n)$ rounds using $O(\log k + \log n)$ space per process.*

4.3. Optimality of the k-clustering in trees

In this subsection, we show that the set *Dom* of clusterheads computed by $\mathcal{CLR}(k)$ has the *minimum* cardinality, for any tree T.

Lemma 19. *Let p any process satisfying $p.\alpha < k$ in a terminal configuration γ of $\mathcal{CLR}(k)$, every child q of p satisfies $q.\alpha \neq k$.*

Proof. Assume the contrary. Then, $MinATall(p) = k$. So:

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

Hence, we obtain a contradiction and consequently $q.\alpha \neq k$. □

Lemma 20. *In any terminal configuration γ of $\mathcal{CLR}(k)$, for every process p, for every process q in $(T(p) \cap Dom) \setminus \{p\}$, we have:*

- *If $p.\alpha \leq k$, then $\|p,q\| > |p.\alpha - k|$.*
- *If $p.\alpha > k$, then $\|p,q\| \geq |p.\alpha - k|$.*

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

If $\texttt{Level}(p) = H$, then p is a leaf and $(T(p) \cap Dom) \setminus \{p\} = \emptyset$, so the lemma trivially holds.

Assume the lemma holds for every process x such that $l < \texttt{Level}(x) \leq H$ and let p be a process such that $\texttt{Level}(p) = l$. Let $q \in (T(p) \cap Dom) \setminus \{p\}$. We have two cases:

q **is a child of** p**:** So, $\|p,q\| = 1$. By definition, $q \in Dom$ in γ. Moreover, as q is not the root, $q.\alpha = k$ in γ by definition of $\mathcal{CLR}(k)$. Then, by Lemma 19, $p.\alpha \geq k$ in γ and we consider two subcases:

$p.\alpha = k$ **in** γ**:** Then, $|p.\alpha - k| = 0$ and the lemma holds.
$p.\alpha > k$ **in** γ**:** Then $p.\alpha = MinATall(p) + 1$ and $MinATall(p) = k$ (because $q.\alpha = k$). So, $p.\alpha = k + 1 \geq 1$ and the lemma holds.

q **is not a child of** p **in** γ**:** Then, there is a child y of p such that $q \in (T(y) \cap Dom) \setminus \{y\}$ in γ (note that $\texttt{Level}(y) = l + 1$). Consider the three following cases:

- $p.\alpha < k$ *in* γ. In this case, $y.\alpha \neq k$ by Lemma 19. So, we consider the two following subcases:
  - $y.\alpha < k$ *in* γ. By the induction hypothesis, we have:

$$\begin{array}{l} \|y,q\| > |y.\alpha - k| \\ \|p,q\| > |y.\alpha - k| + 1 \\ \|p,q\| > |MaxAShort(p) - k| + 1 \\ \|p,q\| > |MaxAShort(p) - (k+1)| \\ \|p,q\| > |MaxAShort(p) + 1 - (k+2)| \\ \|p,q\| > |p.\alpha - (k+2)| \\ \|p,q\| > |p.\alpha - k| \end{array}$$

  - $y.\alpha > k$ *in* γ. Then:

$$\begin{array}{lll} MaxAShort(p) + MinATall(p) & > 2k-2 & \\ MaxAShort(p) - k + 1 & > k - 1 - MinATall(p) & \\ p.\alpha - k & > k - 1 - MinATall(p) & \\ |k - 1 - MinATall(p)| & > |p.\alpha - k| & \\ |k - MinATall(p)| + 1 & > |p.\alpha - k| & \\ |k - y.\alpha| + 1 & > |p.\alpha - k| & \\ |y.\alpha - k| + 1 & > |p.\alpha - k| & \\ \|y,q\| + 1 & > |p.\alpha - k| & \text{(by the induction hypothesis)} \\ \|p,q\| & > |p.\alpha - k| & \end{array}$$

- $p.\alpha = k$ *in* γ. Then, $|p.\alpha - k| = 0$ and as every proper descendant of p is at least at distance 1 from p, the lemma trivially holds.

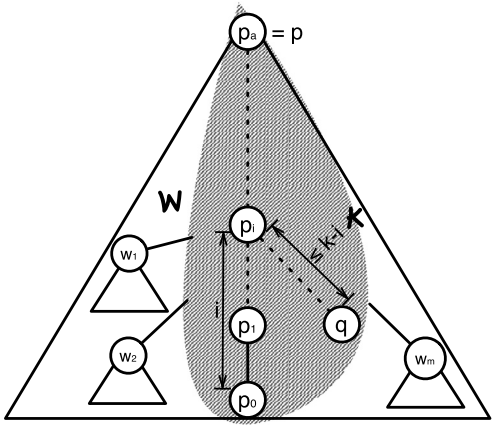

Fig. 6. Illustration of the proof of Theorem 5.

- $p.\alpha > k$ *in* γ. So, we consider the two following subcases:
  - $y.\alpha < k$.

$$\begin{array}{lll}
MaxAShort(p) + MinATall(p) & \leq 2k-2 & \\
MaxAShort(p) + MinATall(p) + 1 & \leq 2k-1 & \\
MaxAShort(p) + p.\alpha & \leq 2k-1 & \\
p.\alpha - k & \leq k - MaxAShort(p) - 1 & \\
|p.\alpha - k| & \leq |k - MaxAShort(p) - 1| & \\
|p.\alpha - k| & \leq |k - MaxAShort(p)| + 1 & \\
|p.\alpha - k| & \leq |MaxAShort(p) - k| + 1 & \\
|p.\alpha - k| & \leq |y.\alpha - k| + 1 & \\
|p.\alpha - k| & \leq \|y, q\| + 1 & \text{(by the induction hypothesis)} \\
\|p, q\| & \geq |p.\alpha - k| &
\end{array}$$

  - $y.\alpha \geq k$. By the induction hypothesis, we have:

$$\begin{array}{l}
\|y, q\| \geq |y.\alpha - k| \\
\|p, q\| \geq |y.\alpha - k| + 1 \\
\|p, q\| \geq |MinATall(p) - k| + 1 \\
\|p, q\| \geq |MinATall(p) + 1 - k| \\
\|p, q\| \geq |p.\alpha - k| \qquad \square
\end{array}$$

Fig. 6 illustrates the proof of the theorem given below.

Theorem 5. *The set Dom of clusterheads computed by* $\mathcal{CLR}(k)$ *is a minimum cardinality k-dominating set of T.*

Proof. Consider the set *Dom* of clusterheads defined in some terminal configuration computed by $\mathcal{CLR}(k)$ in T. We proceed by contradiction: Assume that there exists a k-dominating set DS of T such that $|DS| < |Dom|$. Pick a node p of maximum level such that $T(p) \cap Dom$ contains more nodes than $T(p) \cap DS$, *i.e.*:

- $|T(p) \cap Dom| > |T(p) \cap DS|$, and
- $|T(q) \cap Dom| \leq |T(q) \cap DS|$ for any proper descendant q of p in T.

This means, in particular, that $p \in Dom$ but $p \notin DS$. By definition of *Dom*, $p.\alpha \leq k$. By Property 3.(a), there exists a sequence of nodes $p_0, p_1, \ldots, p_a$, for $a = p.\alpha$, such that:

- $p_a = p$,
- the parent of p_i in T is p_{i+1}, for all $0 \leq i < a$, and
- $p_i.\alpha = i$, for all $0 \leq i \leq a$.

Let $\mathcal{K}$ be the set of all nodes within k hops of p_0.

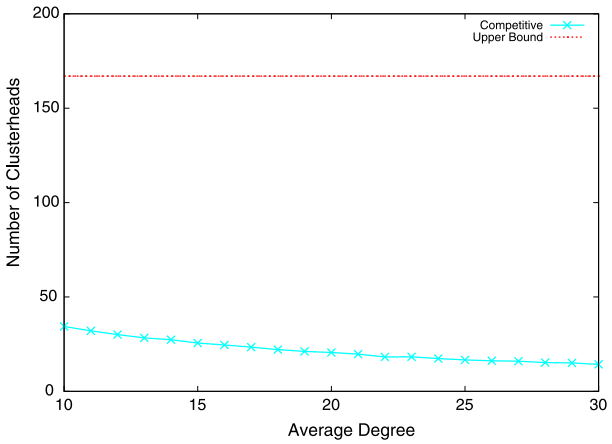

Fig. 7. Competitive *v.s.* its theoretical bound $\lceil \frac{n}{k+1} \rceil$, for $n = 1000$, $k = 5$, and a square field of size $4000m$.

Claim I. $\mathcal{K}$ is a subset of $T(p)$.
Proof of Claim I. If p is the root of T, then the claim trivially holds. Otherwise, $a = p.\alpha = k$, which implies that p_0 is k hops below p, and thus the claim holds.

Claim II. $\mathcal{K} \cap Dom = \{p\}$.
Proof of Claim II. Suppose $q \in \mathcal{K}$ and $q \neq p$. Pick the node p_i that is closest to q. Then, q is at most $k - i$ (*i.e.*, $|p_i.\alpha - k|$) hops below p_i. By Lemma 20, $q \notin Dom$.

Let $\mathcal{W} = T(p) \setminus \mathcal{K}$. Then, $\mathcal{W}$ is the exact union of subtrees rooted at $w_1, w_2, \ldots, w_m$, namely the nodes not in $\mathcal{K}$ whose parents are in $\mathcal{K}$.

Each w_i is a proper descendant of p, and thus, by hypothesis, *DS* must have at least as many members as *Dom* in $\mathcal{W}$. Since *DS* has fewer members than *Dom* in $T(p)$, then *DS* must have fewer members than *Dom* in $\mathcal{K}$. By Claim II, $\mathcal{K} \cap DS = \emptyset$. This implies that *DS* contains no node within k hops of p_0, contradicting the hypothesis that *DS* is a k-dominating set. □

4.4. Experimental results

We ran simulations to study the average performance of our algorithm $(\mathcal{CLR}(k) \circ \mathcal{MIST} \circ \mathcal{BFST})$ in terms of number of clusterheads. For sake of simplicity, our algorithm will be named `Competitive` in this section.

We obtain our experimental results using an event-driven simulator for wireless sensor networks, called *Sinalgo*. In this simulator, processes are randomly deployed on a square plane. Processes are motionless and equipped with radio. Two processes can communicate if and only if their Euclidean distance is at most *rad*, where *rad* is the transmission range. So, the network topology is a Unit Disk Graph (UDG).

We considered connected UDG networks of $n = 1000$ nodes deployed using a uniform random distribution on a $4000m$-side square. We tune the transmission range to control the average degree $\overline{\delta}$ of the network. We vary $\overline{\delta}$ from 10 to 30 and k from 3 to 5. For each setting, the average number of clusterheads is computed over 50 connected UDGs, randomly generated.

We only presented here the results obtained with $k = 5$. However, the general trends observed for $k = 5$ are representative: they can be also observed in other cases we experimented (that is, $k = 3$ and $k = 4$).

We first compared the average performance of our algorithm, `Competitive`, against the theoretical bound proven in Theorem 3. The experimental results are given in Fig. 7. They confirm that `Competitive` is well-suited for wireless sensor networks, since its average performance is drastically better than the theoretical bound, which holds for all arbitrary connected graphs. Note also that, the number of clusterheads decreases when the average degree increases because the diameter of the network also decreases in that case. (This trend can be also observed in all other curves.)

Then we implemented algorithms given in [6] and [8]. To the best of our knowledge, these are the only self-stabilizing algorithms that guarantee a bound on the number of clusterheads. The algorithm given in [8] is a hierarchical composition of three layers. The two first layers, denoted by `DSK` in the following, consists of a spanning tree construction and an algorithm that uses the tree structure to compute a k-dominating set D of at most $\lceil \frac{n}{k+1} \rceil$ processes. The third layer consists of an algorithm that makes D minimal, that is, it computes a minimal k-dominating set that is a subset of D. Note that, experiments in [8] show that best results are obtained using a BFS tree algorithm as first layer. Hence, we do the same here. In the following, we denote by `DSK+` the three layer algorithm.

The minimization module used in [8] is actually the algorithm given in [6]. This algorithm, called `Minimal` in the following, can be used without input, *i.e.*, it can be used independently and directly on a network to compute an unconstrained minimal k-dominating set, whose size is at most $\max(1, n/\lceil \frac{k+1}{2} \rceil)$. We can also compose our algorithm with `Minimal`. This version is denoted by `Competitive+` in the following. We recall the main features of each algorithm in Table 1.

Table 1
Features of each algorithm.

Algorithm	Memory Requirement	Round Complexity	Upper Bound	Minimal ?
`Competitive`	$O(\log k + \log n)$	$O(n)$	$\lceil \frac{n}{k+1} \rceil$	No
`DSK` [8]	$O(\log k + \log n + k \log \frac{N}{k})$	$O(n)$	$\lceil \frac{n}{k+1} \rceil$	No
`Minimal` [6]	$O(\log k + \log n)$	$O(n)$	$\max(1, n/\lceil \frac{k+1}{2} \rceil)$	Yes
`Competitive+`	$O(\log k + \log n)$	$O(n)$	$\lceil \frac{n}{k+1} \rceil$	Yes
`DSK+`	$O(\log k + \log n + k \log \frac{N}{k})$	$O(n)$	$\lceil \frac{n}{k+1} \rceil$	Yes

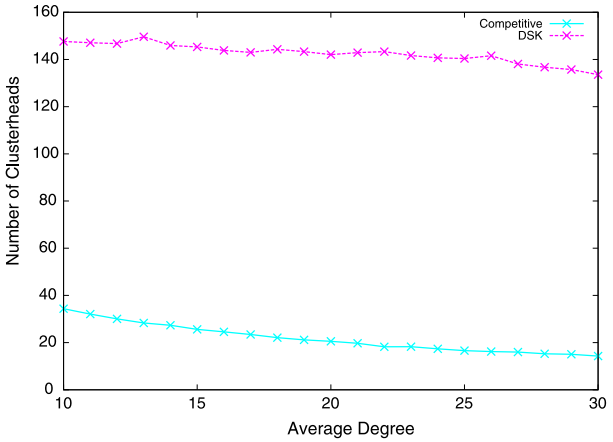

Fig. 8. `Competitive` *v.s.* `DSK`, for $n = 1000$, $k = 5$, and a square field of size $4000m$.

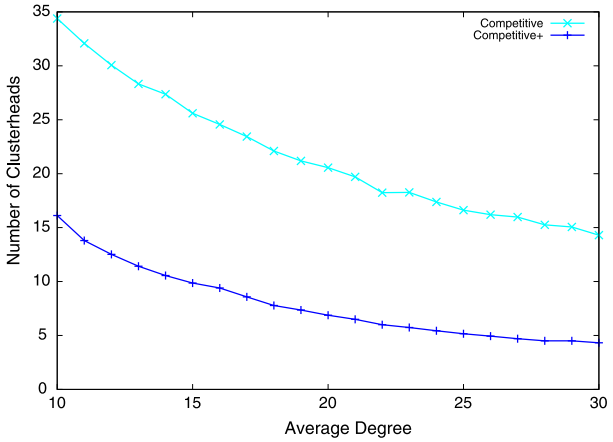

Fig. 9. `Competitive` *v.s.* `Competitive+`, for $n = 1000$, $k = 5$, and a square field of size $4000m$.

We first compared in Fig. 8 the two algorithms that computes a dominating set that is not necessarily minimal, *i.e.* our algorithm `Competitive` and Algorithm `DSK`. Results clearly show that `Competitive` computes notably smaller k-dominating sets than `DSK`.

We then compared `Competitive` and `Competitive+` to see if the minimization really impacts the result. As we can see in Fig. 9, the minimization drastically reduced the size of the computed k-dominating sets.

Finally, Fig. 10 presents results to compare the best version of our algorithm (`Competitive+`) to other algorithms that compute minimal k-dominating sets, that is, `DSK+` and `Minimal`. We can remark that results are really close, but still our algorithm offers the best performances.

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 21, that the cardinality of the MIS computed by $\mathcal{MIST} \circ \mathcal{BFST}$ is bounded by a multiple of the minimum cardinality of any

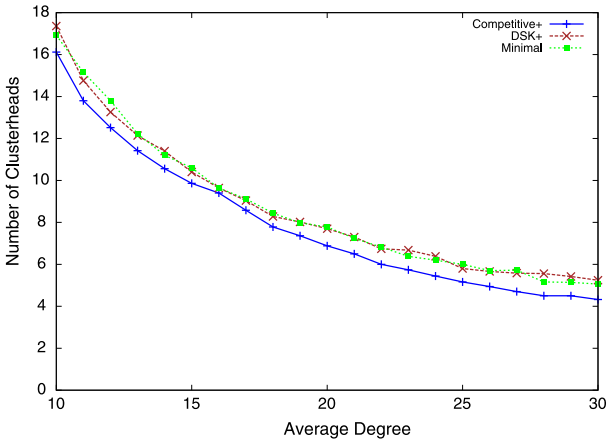

Fig. 10. `Competitive+` *v.s.* `DSK+` *v.s.* `Minimal`, for $n = 1000$, $k = 5$, and a square field of size $4000m$.

k-clustering, then in Lemma 22, 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 multiple of that same minimum.

The proof of Lemma 21 makes use of the following result by Folkman and Graham [21].

Theorem 6 *([21]). Let X be a compact convex region of the plane and $J \subseteq X$ 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.*

Lemma 21. *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.*

Proof. Consider any independent set I and any optimum k-clustering *Opt* of some 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. As the distance between any two distinct members of I is greater than 1, we can apply Theorem 6, that is, no more than $\left(\frac{2}{\sqrt{3}}(\pi k^2) + \frac{1}{2}(2\pi k) + 1\right)$ processes of I can be 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 I is at most $\left(\frac{2\pi k^2}{\sqrt{3}} + \pi k + 1\right) \times |Opt|$. □

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 22. *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 124), every k-cluster of *Clr* contains a path of $k+1$ processes (*i.e.*, of length k), except for the k-cluster which contains r. Since *Clr* is built on T_{MIS}, by Property 1 (page 114), 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:

$$\begin{aligned}
(|Clr| - 1) \times \lceil \tfrac{k}{2} \rceil &\leq |I \setminus \{r\}| \\
(|Clr| - 1)\tfrac{k}{2} &\leq |I| - 1 \\
|Clr| - 1 &\leq \tfrac{2}{k}(|I| - 1) \\
|Clr| &\leq 1 + \tfrac{2}{k}(|I| - 1) \qquad \square
\end{aligned}$$

By Lemmas 21 and 22, we deduce that $|Clr| \leq 1 - \frac{2}{k} + \left(\frac{4\pi k}{\sqrt{3}} + 2\pi + \frac{2}{k}\right)|Opt|$, and since $\frac{4\pi}{\sqrt{3}} \approx 7.2552$, we can claim:

Theorem 7. *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.*

Quasi-unit disk graphs If V is a set of points in the plane, and $\lambda \geq 1$, then we say that $G = (V, E)$ is an *Quasi-Unit Disk Graph (QUDG)* [22] 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 [23]. It is also known as *Approximate Disk Graphs*.

Theorem 8. *For every connected QUDG 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 21, we make use of Theorem 6, but we then consider the surrounding disk of radius λk centered at any clusterhead of an optimum k-clustering 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 QUDG 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 22 and since $\frac{4\pi}{\sqrt{3}} \approx 7.2552$, we are done. □

6. $\mathcal{P}$-Completeness of our MIS construction

The time bottleneck of our k-clustering solution is the MIS tree construction. Indeed, our algorithm $\mathcal{MIST} \circ \mathcal{BFST}$ builds an MIS tree in $\Theta(n)$ rounds in the worst case and, once the MIS Tree is built, the k-clustering is computed in $O(\mathcal{D})$ rounds by Theorem 3 (page 124) and Property 2 (page 117). 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. Indeed, we show below that finding the lexically first MIS tree is $\mathcal{P}$-complete. Since there are networks where $\mathcal{D} = O(\log n)$, this implies that there cannot be an $O(\mathcal{D})$ time distributed algorithm to find the lexically first MIS tree, unless $\mathcal{NC} = \mathcal{P}$.

Whether it would be easier to find an MIS tree, without the restriction that it be lexically first, is still an open question.

$\mathcal{P}$-Completeness of the LFMIS problem with a unique local minimum Given a network $G = (V, E)$, Algorithm $\mathcal{MIST} \circ \mathcal{BFST}$ computes an MIS of G, with respect to the ordering $\prec$ defined in Subsection 3.2. Note that there is a natural lexical ordering on the subsets of V, obtained by writing each subset as a list of processes ordered by $\prec$. The MIS computed by our algorithm comes first in this natural lexical ordering of subsets of V, it is thus the *lexically first* maximal independent set of G.

Let $p_1, \ldots, p_n$ denote the processes of G, ordered by $\prec$. $\mathcal{MIST} \circ \mathcal{BFST}$ takes advantage of an additional property of $\prec$: 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 (Lemma 4, page 116).

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 [24] to be $\mathcal{P}$-complete. However, $\mathcal{MIST} \circ \mathcal{BFST}$ 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 in exhibiting a method to $\mathcal{NC}$-reduce any instance of the *Circuit Value* (CV) problem to an instance of the LFMIS problem with unique local minimum. The CV problem has been shown to be $\mathcal{P}$-complete in [25].

A *Boolean circuit* is a straight line program consisting of finitely many assignments of the form

- $x_i \leftarrow \mathit{true}$,
- $x_i \leftarrow \mathit{false}$,
- $x_i \leftarrow x_j \wedge x_k$ with $j, k < i$,
- $x_i \leftarrow x_j \vee x_k$ with $j, k < i$, or
- $x_i \leftarrow \neg x_j$ with $j < i$,

where each variable x_i in the program appears on the left side of exactly one assignment. The conditions $j, k < i$ and $j < i$ ensure acyclicity. (This implies in particular that the right side of the first assignment is a constant *true* or *false*). The CV problem is then defined to be the evaluation the value of variable x_n in such a program, where n is the maximum index. An example of such a program is given in Fig. 11a. The program can be also represented using logic gates, see in Fig. 12a.

We now exhibit a method to $\mathcal{NC}$-reduce any instance of the $\mathcal{P}$-complete CV problem to an equivalent instance of the LFMIS problem with unique local minimum, in order to prove that the LFMIS problem with unique local minimum is $\mathcal{P}$-complete. First, we show in Lemma 23 that every Boolean circuit program can be expressed into an intermediate equivalent form called *paired form* (defined in Definition 3). Next, in the proof of Theorem 9, we consider any Boolean circuit written in paired form. We transform this circuit into another intermediate *reduced form*, from which it is easy to finally obtain an equivalent instance of the LFMIS problem with unique local minimum. We show that each of these three transformations can be computed in polylogarithmic time using a polynomial number of processes.

The example of Boolean circuit given in Fig. 12a is actually in paired form. Its reduced form is given in Fig. 11b. Fig. 12 represents the same circuits with logic gates. In Fig. 13, we show an equivalent instance of the LFMIS problem with unique local minimum, which is the result of the transformation given at the end of the proof of Theorem 9.

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

Fig. 11. (a) A Boolean circuit in paired form, (b) its reduced form, and (c) the correspondence between variables of both circuits.

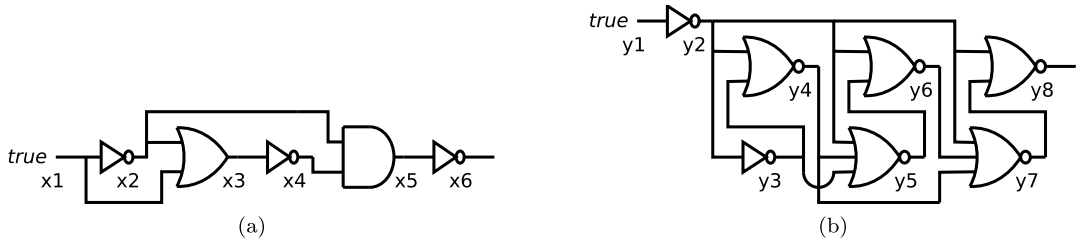

Fig. 12. (a) The same Boolean circuit, and (b) its reduced form using Logic gates.

Definition 3 *(Paired form).* A Boolean circuit is said to be in *paired form* if the number n of its variables is even and for every $i \in [1..n]$, we have:

- If i is even, then the right side of the i^{th} assignment is the negation of the $(i-1)^{\text{th}}$ assigned variable.
- If i is odd, then the right side of the i^{th} assignment is either a constant or the conjunction or disjunction of two prior variables.

Lemma 23. *Any Boolean circuit can be rewritten into an equivalent Boolean circuit in paired form, in constant time using a polynomial number of processes in parallel.*

Proof. Consider any Boolean circuit containing n variables. Recall that x_i denotes the i^{th} assigned variable of the circuit. Here, a, b, c, and d denote new variables. Apply the following transformation on each of the n assignments.

- *If the i^{th} assignment at even rank is not $\neg x_{i-1}$.* Then, we have two cases:
  - $i \neq n$*:* Insert $a \leftarrow \neg x_{i-1}$ and $b \leftarrow \neg x_i$ respectively before and after that assignment.
  - $i = n$*:* We have to ensure that the output of the circuit remains unchanged. Insert $a \leftarrow \neg x_{i-1}$ before the i^{th} assignment and insert the assignments $b \leftarrow \neg x_i$, $c \leftarrow b \wedge b$, and $d \leftarrow \neg c$ after the i^{th} assignment. Then, the new output will be $d = \neg c = \neg(b \wedge b) = \neg b = \neg\neg x_i = x_i$.

  In both cases the truth value of every variable x_k with $k \in [1..n]$ remains unchanged.
- *If the i^{th} assignment at odd rank is a negation $x_i \leftarrow \neg x_j$ with $j < i$ and $i < n$.* Then, replace the i^{th} assignment by $a \leftarrow x_j \vee x_j$, $b \leftarrow \neg a$ and $x_i \leftarrow b \vee b$. In particular, after the transformation, we have $x_i = b \vee b = \neg a \vee \neg a = \neg(x_j \vee x_j) \vee \neg(x_j \vee x_j) = \neg x_j \vee \neg x_j = \neg x_j$. The truth value of every variable x_k with $k \in [1..n]$ remains unchanged.
- *If the n^{th} assignment is at an odd rank.* Then, we should add assignments so that the number of assignments of the new circuit becomes even. Moreover, we have to ensure that the output of the circuit remains unchanged. We have two cases:
  - *The assignment is a negation $x_n \leftarrow \neg x_j$ with $j < n$.* Replace the n^{th} assignment by $a \leftarrow x_j \vee x_j$ and $x_n \leftarrow \neg a$. Then, the output remains unchanged since $x_n = \neg a = \neg(x_j \vee x_j) = \neg x_j$.
  - *The assignment is not a negation.* Add assignments $a \leftarrow \neg x_n$, $b \leftarrow a \wedge a$, and $c \leftarrow \neg b$ at the end of the circuit. The new output will be $c = \neg b = \neg(a \wedge a) = \neg a = \neg\neg x_n = x_n$.

  In both cases the truth value of every variable x_k with $k \in [1..n]$ remains unchanged.

After the transformation, we obtain a Boolean circuit of the paired form. The value of the last variable of this circuit is the same as that of the last variable of the initial circuit. Finally, note that there are $O(n)$ transformations. Each transformation is independent of all others, and hence can be done in constant time. Thus, the whole transformation can be done in constant time using a polynomial number of processes in parallel. □

Theorem 9. *The LFMIS problem with unique local minimum is $\mathcal{P}$-complete.*

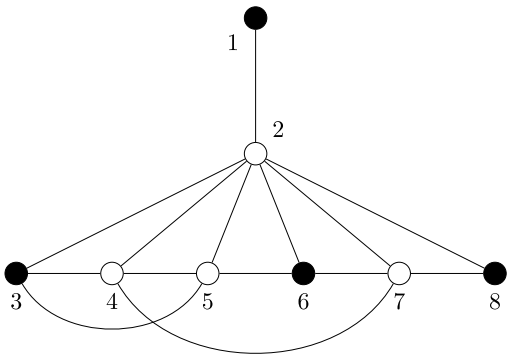

Fig. 13. Resulting instance of the LFMIS problem.

Proof. Consider an instance of CV problem, that is a Boolean circuit. Recall that x_i denotes the i^{th} assigned variable of the circuit. Without loss of generality, we can assume that this instance is in the paired form. Indeed, this assumption can be enforced using the $\mathcal{NC}$-reduction given in Lemma 23. Thus, from Definition 3, assuming an even number of variables, we note them $x_1, x_2, \ldots, x_{2n}$. For any $i \in [1..n]$, refer to x_{2i-1} and x_{2i} as *partners*. Note that partners always take opposite Boolean values when evaluated.

The rest of the proof is divided into two parts. We first $\mathcal{NC}$-reduce the initial instance of the CV problem into an intermediate *reduced form* (i). Then, we transform that reduced form of the circuit into an equivalent instance of the LFMIS problem with unique local minimum (ii).

(i) *Reduced form.* Rewrite the circuit into reduced form, where the variables are $y_1, y_2, \ldots, y_{2n+2}$. The first assignment will be $y_1 \leftarrow \mathit{true}$, and the second assignment will be $y_2 \leftarrow \neg y_1$. There will be a one-to-one correspondence between the variables of the initial circuit and all but the first two variables of the circuit in the reduced form: For any $i \in [1..n]$, 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. This order will be solved by the rewriting, allowing in particular to know which of y_{2n+1} and y_{2n+2} corresponds to x_{2n}, the output of the initial circuit. Thus, y_{2i+1} and y_{2i+2} will also have opposite values and we will also refer to these variables as *partners*. We use the following rewriting rules to construct the reduced form of the circuit, for any $i \in [1..n]$.

1. The $(2i+2)^{\text{nd}}$ assignment of the reduced circuit will be $y_{2i+2} \leftarrow \neg y_2 \wedge \neg y_{2i+1}$. That is, y_{2i+2} is assigned the Boolean value opposite to that of its odd partner y_{2i+1}, since $\neg y_2 = \mathit{true}$.
2. The $(2i+1)^{\text{st}}$ assignment of the reduced circuit will depend on the $(2i-1)^{\text{st}}$ assignment in the initial circuit:
   (a) If the $(2i-1)^{\text{st}}$ assignment of the initial circuit is $x_{2i-1} = \mathit{true}$, then the $(2i+1)^{\text{st}}$ assignment of the reduced circuit will be $y_{2i+1} \leftarrow \neg y_2$ (that is, *true*). Thus, y_{2i+1} will correspond to x_{2i-1}, and y_{2i+2} will correspond to x_{2i}.
   (b) If the $(2i-1)^{\text{st}}$ assignment of the initial circuit is $x_{2i-1} = \mathit{false}$, then the $(2i+1)^{\text{st}}$ assignment of the reduced circuit will be $y_{2i+1} \leftarrow \neg y_2$ (that is, *true*). Thus, y_{2i+1} will correspond to x_{2i}, and y_{2i+2} will correspond to x_{2i-1}.
   (c) If the $(2i-1)^{\text{st}}$ assignment of the initial circuit 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 $(2i+1)^{\text{st}}$ assignment of the reduced circuit will be $y_{2i+1} \leftarrow \neg y_2 \wedge \neg y_p \wedge \neg y_q$ (that is, $\mathit{true} \wedge \neg\neg x_j \wedge \neg\neg x_k = x_j \wedge x_k$). Thus, y_{2i+1} will correspond to x_{2i-1}, and y_{2i+2} will correspond to x_{2i}.
   (d) If the $(2i-1)^{\text{st}}$ assignment of the initial circuit 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 $(2i+1)^{\text{st}}$ assignment of the reduced circuit will be $y_{2i+1} \leftarrow \neg y_2 \wedge \neg y_p \wedge \neg y_q$ (that is, $\mathit{true} \wedge \neg(y_p \vee y_q) = \neg(x_j \vee x_k)$). Thus, y_{2i+1} will correspond to x_{2i}, and y_{2i+2} will correspond to x_{2i-1}.

By construction, the partner variables of the reduced circuit will always be assigned opposite truth values. Through simple induction, we can see that evaluation of the reduced circuit will assign *true* to y_1, *false* to y_2, and to each variable of the reduced circuit the same value as the corresponding variable in the initial circuit.

(ii) *Equivalent instance of LFMIS problem.* Finally, we construct an equivalent instance of the LFMIS problem with unique local minimum as follows. Let G be the network whose ordered (*w.r.t.* UIDs) list of processes is $p_1, p_2, \ldots, p_{2n+2}$, 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 reduced circuit. The LFMIS problem with unique local minimum for the reduced circuit described in Fig. 11b and represented using Logic gates in Fig. 12b is shown in Fig. 13. We remark that the distances of all processes to p_1 are: $\|p_1, p_1\| = 0$, $\|p_2, p_1\| = 1$, and $\forall 2 < i \leq 2n+2$, $\|p_i, p_1\| = 2$. Consequently, for every $1 < i \leq 2n+2$, $p_{i-1} \prec p_i$.

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, which implies that,

for all $3 \leq i \leq 2n+2$, local computation of the LFMIS at process p_i depends only on the computation at prior processes $p_2, \ldots, p_{i-1}$.

By simple induction on process ordering, we can see that $p_i \in I$ if and only if y_i, and hence its corresponding variable of the initial circuit, are assigned the value *true*.

Note that all steps of the reduction can be accomplished in parallel in polylogarithmic time with polynomially many processors. Thus, any instance of CV problem can be $\mathcal{NC}$-reduced to an instance of the LFMIS problem with unique local minimum. □

7. Conclusion and perspectives

We have given a silent self-stabilizing algorithm for constructing a k-clustering of any asynchronous connected network with unique IDs. Our algorithm stabilizes in $O(n)$ rounds, using $O(\log k + \log n)$ space per process, where n is the number of processes. Our algorithm is uniform in the sense that it does not require processes to know any upper bound on the size n or the diameter $\mathcal{D}$ of the network. This is the first algorithm of k-clustering construction that is both self-stabilizing and competitive in UDG and QUDG networks. Moreover, in case of tree networks, our algorithm computes an optimal k-clustering.

An immediate extension of this work would be to sharpen the competitive 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.K. Datta, L.L. Larmore, S. Devismes, K. Heurtefeux, Y. Rivierre, Competitive self-stabilizing k-clustering, in: IEEE 32nd International Conference on Distributed Computing Systems, ICDCS, Macao, China, 18–21 June 2012, IEEE, 2012, pp. 476–485.
[2] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, 1979.
[3] E.W. Dijkstra, Self-stabilizing systems in spite of distributed control, Commun. ACM 17 (1974) 643–644.
[4] E. Caron, A.K. Datta, B. Depardon, L.L. Larmore, A self-stabilizing k-clustering algorithm for weighted graphs, J. Parallel Distrib. Comput. 70 (11) (2010) 1159–1173.
[5] A.K. Datta, L.L. Larmore, P. Vemula, A self-stabilizing $O(k)$-time k-clustering algorithm, Comput. J. (2009) bxn071.
[6] A.K. Datta, S. Devismes, L.L. Larmore, A self-stabilizing $O(n)$-round k-clustering algorithm, in: SRDS, 2009, pp. 147–155.
[7] E. Caron, A.K. Datta, B. Depardon, L.L. Larmore, A self-stabilizing k-clustering algorithm for weighted graphs, J. Parallel Distrib. Comput. 70 (11) (2010) 1159–1173.
[8] A.K. Datta, S. Devismes, K. Heurtefeux, L.L. Larmore, Y. Rivierre, Self-stabilizing small k-dominating sets, in: ICNC, IEEE Computer Society, 2011, pp. 30–39.
[9] A.D. Amis, R. Prakash, D. Huynh, T. Vuong, Max–min D-cluster formation in wireless ad hoc networks, in: IEEE INFOCOM, 2000, pp. 32–41.
[10] Y. Fernandess, D. Malkhi, K-clustering in wireless ad hoc networks, in: POMC, 2002, pp. 31–37.
[11] M.A. Spohn, J.J. Garcia-Luna-Aceves, Bounded-distance multi-clusterhead formation in wireless ad hoc networks, Ad Hoc Netw. 5 (2004) 504–530.
[12] V. Ravelomanana, Distributed k-clustering algorithms for random wireless multihop networks, in: ICN, 2005, pp. 109–116.
[13] K.M. Alzoubi, P. Wan, O. Frieder, New distributed algorithm for connected dominating set in wireless ad hoc networks, in: 35th Hawaii International Conference on System Sciences, HICSS-35, Big Island, HI, USA, 7–10 January 2002, IEEE Computer Society, 2002, p. 297, CD-ROM/abstracts proceedings.
[14] S. Dolev, Self-Stabilization, MIT Press, 2000.
[15] S. Dolev, A. Israeli, S. Moran, Uniform dynamic self-stabilizing leader election, IEEE Trans. Parallel Distrib. Syst. 8 (1997) 424–440.
[16] S. Dolev, M.G. Gouda, M. Schneider, Memory requirements for silent stabilization, in: PODC, 1996, pp. 27–34.
[17] G. Tel, Introduction to Distributed Algorithms, 2nd edn., Cambridge University Press, 2001.
[18] S.A. Cook, Deterministic CFL's are accepted simultaneously in polynomial time and log squared space, in: STOC, ACM, 1979, pp. 338–345.
[19] S.T. Huang, N.S. Chen, A self-stabilizing algorithm for constructing breadth-first trees, Inform. Process. Lett. 41 (1992) 109–117.
[20] A.K. Datta, L.L. Larmore, P. Vemula, An o(n)-time self-stabilizing leader election algorithm, J. Parallel Distrib. Comput. 71 (11) (2011) 1532–1544.
[21] J.H. Folkman, R.L. Graham, A packing inequality for compact convex subsets of the plane, Canad. Math. Bull. 12 (6) (1969) 745–752.
[22] F. Kuhn, R. Wattenhofer, A. Zollinger, Ad-hoc networks beyond unit disk graphs, in: DIALM–POMC, ACM, 2003, pp. 69–78.
[23] L. Barrière, P. Fraigniaud, L. Narayanan, Robust position-based routing in wireless ad hoc networks with unstable transmission ranges, in: DIALM, ACM, 2001, pp. 19–27.
[24] S. Cook, A taxonomy of problems with fast parallel algorithms, Inform. Control 64 (1985) 2–22.
[25] R.E. Ladner, The circuit value problem is log space complete for P, SIGACT News 7 (1975) 18–20.