A Self-Stabilizing $O(k)$-Time K-Clustering Algorithm

Ajoy K. Datta* Lawrence L. Larmore
Priyanka Vemula
School of Computer Science, University of Nevada Las Vegas

Abstract

A silent self-stabilizing asynchronous distributed algorithms is given for constructing a k-dominating set, and hence a k-clustering, of a connected network of processes with unique IDs and no designated leader. The algorithm is comparison-based, takes $O(k)$ time and uses $O(k \log n)$ space per process, where n is the size of the network.

It is known that finding a minimal k-dominating set is $\mathcal{NP}$-hard. A lower bound is given, showing that no comparison-based algorithm for the k-clustering problem that takes $o(diam)$ rounds can approximate the optimal solution, where $diam$ is the diameter of the network.

Keywords: K-clustering, self-stabilization.

1 Introduction

Let $G = (V, E)$ be a connected graph. If $x, y \in V$, we define $||x, y||$, the *distance* from x to y, to be the length of the shortest path from x to y. We also define

$$\begin{aligned} diameter(G) &= \max_{x,y \in V} ||x, y|| \\ radius(G) &= \min_{x \in V} \max_{y \in V} ||x, y|| \end{aligned}$$

Given a non-negative integer k, we define a *k-cluster* of G to be a non-empty subgraph of G of radius at most k. If C is a k-cluster of G, we say that $x \in C$ is a *clusterhead* of C if, for any $y \in C$, there is a path of length at most k in C from x to y.

We define a *k-clustering* of G to be a set $\{C_1, \ldots C_m\}$ of k-clusters of G such that every vertex $x \in V$ lies in exactly one of the C_i. The *k-clustering problem* is the problem of finding a k-clustering of a given graph.[1]

A set of vertices $D \subseteq V$ is a *k-dominating set* of G if, for every $x \in V$, there exists $y \in D$ such that $||x, y|| \leq k$. A k-dominating set gives rise to a k-clustering; for each $x \in V$, let $Leader(x) \in D$ be the member of D that is closest to x. For each $y \in D$, let $C_y = \{x \in V : Leader(x) = y\}$. Then $\{C_y\}_{y \in D}$ is a k-clustering of G. (Ties must be broken carefully to maintain the connectedness of each cluster.) We say that a k-dominating set D is *optimal* if no k-dominating set of G has fewer elements than D. The problem of finding an optimal k-dominating set is known to be $\mathcal{NP}$-hard.

*Contact Author: Ajoy K. Datta. Email: datta@cs.unlv.edu.

[1]There are several alternative definitions of k-clustering, or the k-clustering problem, in the literature.

1.1 Related Work

To the best of our knowledge, there exist only three asynchronous distributed solutions to the k-clustering problem in mobile *ad hoc* networks (MANETs), in the comparison based model, *i.e.,* where the only operation allowed on IDs is comparison. Amis *et al.* [1] give the first distributed solution this problem. The time and space complexities of their solution are $O(k)$ and $O(k \log n)$, respectively. Spohn and Garcia-Luna-Aceves [9] 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 k-clustering problem discussed in this paper is then the case $m = 1$. The time and space complexities of the distributed algorithm in [9] are not given.

Two synchronous distributed algorithm which compute k-dominating sets using a non-comparison based model are given in [6, 7]. A synchronous algorithm for k-clustering for wireless radio multi-hop networks is presented in [8]. Amis *et al.* give a non-self stabilizing message passing algorithm for the k-clustering problem which takes $O(k)$ steps, and requires $(3k+O(1)) \log_2 n$ bits of memory in each process [1].

Fernandess and Malkhi [5] give a non-self stabilizing message passing algorithm for the k-clustering problem that uses $O(\log n)$ memory per process, takes $O(n)$ steps, providing a BFS tree for the network is already given. In the special case that the network is a unit disk graph in the plane, their algorithm is $8k$-competitive, meaning that the number of clusters constructed by their algorithm is at most $(8k + C)$ times the minimum possible number of clusters in a k-clustering of the same network, where C is a constant that depends neither on the network nor on k. The proof of competitiveness given by Fernandess and Malkhi contains a flaw, although their result is correct; they incorrectly state that at most k^2 disjoint disks of radius 1 can be placed in a $2k \times 2k$ square in the plane.

1.2 Contributions

None of the above mentioned solutions is self-stabilizing [2, 3]. Our algorithm, FLOOD, given in Section 3, is similar to that of Amis *et al.* [1]. FLOOD uses only $(2k + O(1)) \log_2 n$ bits per process, approximately half that of [1]. FLOOD finds a k-dominating set in a network of processes, assuming that each process has a unique ID. FLOOD is self-stabilizing and silent, and takes $3k + O(1)$ rounds.

We say that an algorithm is *comparison-based* if the only operation it can use to distinguish two IDs is comparison. FLOOD is comparison-based. In contrast, an algorithm that examines individual bits of an ID is not comparison based. In Section 4, we give a case where FLOOD picks the majority of processes in the network to be clusterheads. This duplicates the bad behavior of Amis *et al.*'s algorithm [1]. In Section 5, we show that this bad case is unavoidable in the comparison model, *i.e.,* that there is no competitive comparison-based self-stabilizing distributed asynchronous algorithm for the k-clustering algorithm which takes $o(diam)$ rounds, even if all processes have unique IDs.

1.3 Outline of the Paper

In Section 2, we describe the model of computation used in the paper, and give some additional needed definitions. In Section 3, we define the algorithm FLOOD, and prove its correctness and time and space complexity. We also show an example execution of FLOOD, using the network shown in Figure 3.1. In Section 4, we give a case where FLOOD performs badly, and in Section

5, we prove that this bad performance is unavoidable for any fast algorithm for the k-clustering problem that uses only comparisons to distinguish IDs. Section 6 concludes the paper.

2 Preliminaries

We are given a connected undirected notwork, $G = (V, E)$ of $|V| = n$ processes, where $n \geq 2$, and a distributed algorithm $\mathcal{A}$ on that network. Each process P has a unique ID, $P.id$, a non-negative integer. We assume the *shared memory model* of computation introduced in [2]. In this model, process P maintains registers. P can read its own registers and those of its neighbors, but can write only to its own registers.

The *state* of a process is defined by the values of its registers. A *configuration* of the network is a function from processes to states; if γ is the current configuration, then $\gamma(P)$ is the current state of each process P. An *execution* of $\mathcal{A}$ is a sequence of states $e = \gamma_0 \mapsto \gamma_1 \mapsto \ldots \mapsto \gamma_i \ldots$, where $\gamma_i \mapsto \gamma_{i+1}$ means that it is possible for the network to change from configuration γ_i to configuration γ_{i+1} in one step. We say that an execution is *maximal* if it is infinite, or if it ends at a *sink*, *i.e.,* a configuration from which no execution is possible.

The *program* of each process consists of a set of registers and a finite set of actions of the following form: $< label >:: < guard > \longrightarrow < statement >$. 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. A step $\gamma_i \mapsto \gamma_{i+1}$ consists of one or more *enabled* processes executing an *action*. The evaluations of all guards and executions of all statements of those actions are presumed to take place in one atomic step; this model is called *composite atomicity* [3].

We assume that each transition from a configuration to another is driven by a *scheduler*, also called a *daemon*. If one or more processes are enabled, the daemon selects at least one of these enabled processes to execute an action. We assume that the daemon is also *unfair*, meaning that, even if a process P is continuously enabled, P might never be selected by the daemon unless P is the only enabled process.

We say that a process P is *neutralized* in the computation 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.

We use the notion of *round* [4], which captures the speed of the slowest process in an execution. We say that a finite execution $\varrho = \gamma_i \mapsto \gamma_{i+1} \mapsto \ldots \mapsto \gamma_j$ is a *round* if the following two conditions hold:

1. Every process P that is enabled at γ_i either executes or becomes neutralized during some step of ϱ.

2. The execution $\gamma_i \mapsto \ldots \mapsto \gamma_{j-1}$ does not satisfy condition 1.

We define the *round complexity* of an execution to be the number of disjoint rounds in the execution, possibly plus 1 if there are some steps left over.

2.1 Self-Stabilization and Silence

The concept of *self-stabilization* was introduced by Dijkstra [2]. Informally, we say that $\mathcal{A}$ is *self-stabilizing* if, starting from a completely arbitrary configuration, the network will eventually reach a legitimate configuration.

More formally, we assume that we are given a *legitimacy predicate* $\mathcal{L}_{\mathcal{A}}$ on configurations. Let $\mathbb{L}_{\mathcal{A}}$ be the set of all *legitimate* configurations, *i.e.,* configurations which satisfy $\mathcal{L}_{\mathcal{A}}$. Then we define $\mathcal{A}$ to be *self-stabilizing* if the following two conditions hold:

1. (Convergence) Every maximal execution contains some member of $\mathbb{L}_{\mathcal{A}}$.
2. (Closure) If an execution e begins at a member of $\mathbb{L}_{\mathcal{A}}$, then all configurations of e are members of $\mathbb{L}_{\mathcal{A}}$.

We say that $\mathcal{A}$ is *silent* if every execution is finite. In other words, starting from an arbitrary configuration, the network will eventually reach a configuration where no process is enabled.

3 The Algorithm FLOOD

In this section, we present a silent, self-stabilizing algorithm FLOOD, which computes a k-clustering of a network. FLOOD uses $O(k \log n)$ space per process, and self-stabilizes within $O(k)$ rounds.

Throughout, we write $\mathcal{N}_P$ for the set of all neighbors of P, and $\mathcal{U}_P = \mathcal{N}_P \cup \{P\}$.

Basic Idea of FLOOD. The basic idea of FLOOD is that a process P is chosen to be a *clusterhead* if and only if, for some process Q, P has the smallest ID of any process within k hops of Q. It requires at most $2k$ rounds for each process to be informed that is, or is not a clusterhead.

A clustering of the network is then obtained by every process joining a tree rooted at the nearest clusterhead; the processes of each tree become one cluster. Every process is within k hops of some clusterhead, and thus our clustering is a k-clustering.

Implementation of FLOOD. Each process P contains two arrays, $P.minid[d]$ for $1 \leq d \leq k$, and $P.maxminid[d]$ for $1 \leq d \leq k$. In addition, P has variables $P.leader$ and $P.parent$, both IDs, and $P.dist$, a non-negative integer. Each of these variables has a *stable value*, namely that value that each will have when FLOOD stabilizes. In Lemma 3.2, we will prove the following:

- The stable value of $P.minid[d]$ is the smallest ID of any process within d hops of P. If $P.minid[k] = P.id$, then P is a clusterhead, however, a process could be a clusterhead without being the smallest ID within k hops of itself. We thus need to compute another array.
- The stable value of $P.maxminid[d]$ is the largest value of $Q.minid[k]$ for any process within within d hops of P. P is a clusterhead if and only if the stable value of $P.maxminid[k]$ is $P.id$.
- The stable value of $P.leader$ is the ID of P's clusterhead, *i.e.,* the clusterhead nearest to P.
- The stable value of $P.dist$ is the distance from P to its clusterhead.

- If P is not a clusterhead, the stable value of $P.parent$ is the ID of the neighbor of P on the shortest path from P to its clusterhead, *i.e.,* the parent of P in the BFS spanning tree of its cluster; that is, $P.leader = P.parent.leader$ and $P.dist = P.parent.dist + 1$.
- The stable values of *leader* define a spanning forest in the network, where the clusterheads are the roots, and the trees are the k-clusters.
- All variables stabilize within $3k + 1$ rounds of arbitrary initialization.

We say that a variable is *consistent* if no action which could alter that variable, or which has a lower priority number, is enabled. We say that a variable is *stable* if all actions which could change the value of that variable are silent. Thus, any stable variable is consistent. The converse does not hold, however. It is possible for a process P and all its neighbors to be initialized in such a way that P is not initially enabled to execute any action, which implies that all variables of P are initially consistent; and yet, in a later round, some variables of some neighbors of P could change in such a way that P is enabled to execute an action. P acts by checking its variables against those of its neighbors. A variable of P is *consistent* if it satisfies the appropriate rule in the list below.

- $P.minid[1] = \min\{Q.id : Q \in \mathcal{U}_P\}$.
- For $d > 1$, $P.minid[d] = \min\{Q.minid[d-1] : Q \in \mathcal{U}_P\}$.
- $P.maxminid[1] = \max\{Q.minid[k] : Q \in \mathcal{U}_P\}$.
- For $d > 1$, $P.maxminid[d] = \max\{Q.maxmin[d-1] : Q \in \mathcal{U}_P\}$.
- If $P.dist = 0$, then $P.parent = P.id$.
- If $P.dist > 0$, then $P.parent = \min\{Q.id : (Q \in \mathcal{N}_P) \wedge (Q.dist + 1 = P.dist)\}$.
- $P.leader = P.parent.leader$.

If all variables of P are consistent, then P is not enabled to execute any action. Otherwise, P will identify the inconsistent variable of lowest priority number, and change its value to make it consistent; where $P.minid[d]$ has priority d, $P.maxmin[d]$ has priority $d+k$, and $P.dist$, $P.parent$ and $P.leader$ each have priority $2k + 1$. To save time, FLOOD changes those last three variables in a single action. When all variables of all processes are consistent, FLOOD is silent.

Resolving Ties. Ties, which occur when a process P is equidistant to two nearest clusterheads, can be resolved arbitrarily. We choose to use the "lowest ID of neighbor" rule: if Q_1 and Q_2 are neighbors of P, where $Q_1.id < Q_2.id$, and if, stably, $P.dist = Q_1.dist + 1 = Q_2.dist + 1$, then the stable value of $P.parent$ might be $Q_1.id$, but cannot be $Q_2.id$.

3.1 Functions and Actions of FLOOD

We now give a formal definition of FLOOD. Each process P has the following variables. Each variable is of ID type, except $P.dist$, which is a non-negative integer.

$P.id$.

$P.minid[d]$ for $1 \leq d \leq k$.

$P.maxminid[d]$ for $1 \leq d \leq k$.

$P.dist$.

$P.parent$.

$P.leader$.

Each process P can evaluate the following functions by reading its variables and those of its neighbors.

$$MinIdF(P,d) = \begin{cases} \min\{Q.id : Q \in \mathcal{U}_P\} & \text{if } d = 1 \\ \min\{Q.minid[d-1] : Q \in \mathcal{U}_P\} & \text{if } 2 \leq d \leq k \end{cases}$$

$$MaxMinIdF(P,d) = \begin{cases} \max\{Q.minid[k] : Q \in \mathcal{U}_P\} & \text{if } d = 1 \\ \max\{Q.maxminid[d-1] : Q \in \mathcal{U}_P\} & \text{if } 2 \leq d \leq k \end{cases}$$

$IsClusterheadF(P) \equiv P.maxminid[k] = P.id$, of Boolean type.

$$DistF(P) = \begin{cases} 0 & \text{if } IsClusterheadF(P) \\ \min\{Q.dist + 1 : Q \in \mathcal{N}_P\} & \text{otherwise} \end{cases}$$

$$ParentF(P) = \begin{cases} P.id & \text{if } IsClusterheadF(P) \\ \min\{Q.id : (Q \in \mathcal{N}_P) \wedge \\ \qquad (Q.dist + 1 = DistF(P))\} & \text{otherwise} \end{cases}$$

$$LeaderF(P) = \begin{cases} P.id & \text{if } IsClusterheadF(P) \\ P.parent.leader & \text{otherwise} \end{cases}$$

The actions of FLOOD are given in Table 1. Each action is given a priority number. Each action's guard includes the condition given in the third column of the action tables, and also includes the condition that no action which has an earlier priority number is enabled. We say that an action becomes *silent* if it will never again be enabled. We say that a module (or a program) *converges* if all its actions become silent.

Table 1: Actions of FLOOD

A1(d) $1 \leq d \leq k$ priority d	Floodmin	$(P.minid[d] \neq MinIdF(P))$	$\longrightarrow$	$P.minid[d] \leftarrow$ $\quad MinIdF(P,d)$
A2(d) $1 \leq d \leq k$ priority $k+d$	Floodmax	$P.maxminid[d] \neq MaxMinIdF(P,d)$	$\longrightarrow$	$P.maxminid[d] \leftarrow$ $\quad MaxMinIdF(P,d)$
A3 priority $2k+1$	Clustering	$(P.dist \neq DistF(P))) \vee$ $(P.parent \neq ParentF(P)) \vee$ $(P.leader \neq LeaderF(P))$	$\longrightarrow$	$P.dist \leftarrow DistF(P)$ $P.parent \leftarrow$ $\quad ParentF(P)$ $P.leader \leftarrow$ $\quad LeaderF(P)$

3.2 An Example Computation

In Figure 3.1, we give a network, which we call the *standard graph*. In Figure 3.2, which consists of 12 subfigures, we illustrate the steps of a computation of FLOOD, where $k = 4$.

Figures 3.2(a) through 3.2(d) show the stable values of $minid[i], i = 1 \ldots 4$, which are computed by (Action A1). Figures 3.2(e) through 3.2(h) show the stable values of $maxminid[i], i = 1 \ldots 4$ which are computed by (Action A2). Figure 3.2(h) also shows the final selection of clusterheads, namely processes 10, 13, and 14. Clusterheads are indicated by larger dots in Figures (h) through (l). Figures 3.2(i) though 3.2(l) demonstrate the growth of clusters around the three clusterheads. Note that the cluster subgraphs are BFS trees rooted at the clusterheads.

We used boxed numbers and dashed polygonal lines to identify the different *zones* created by the $minid[i]$ (in Figures 3.2(a) through 3.2(d)) and $maxminid[i]$

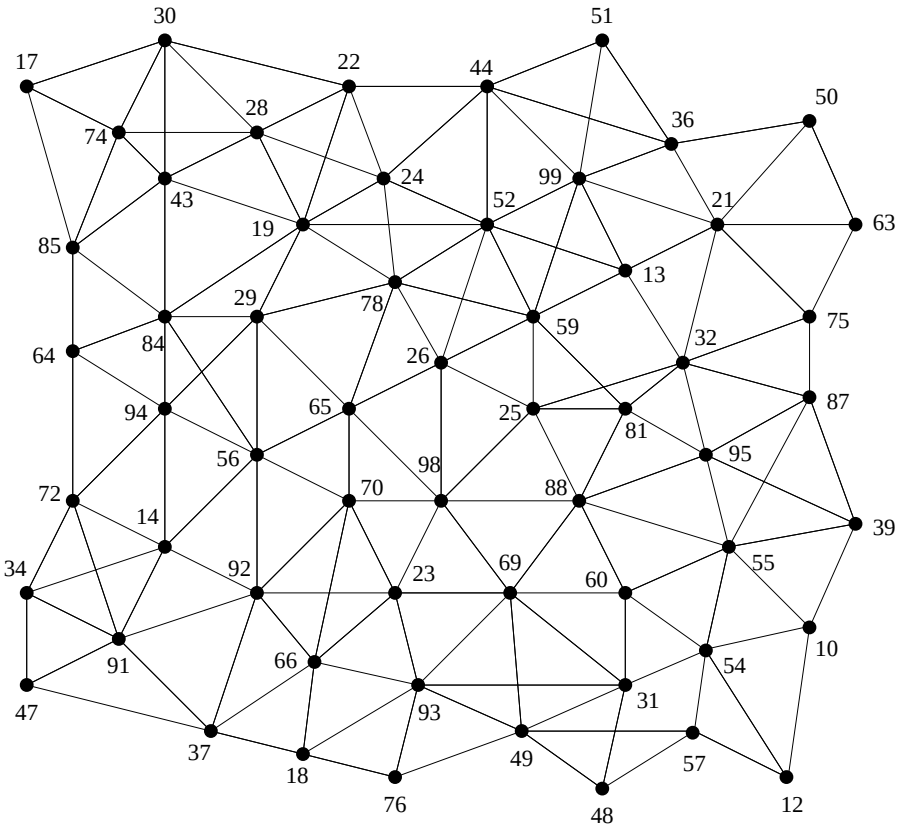
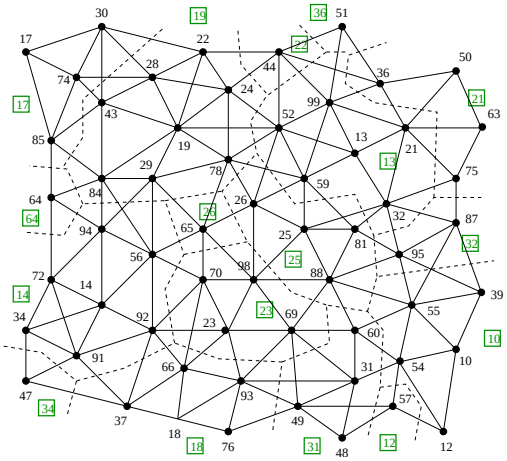


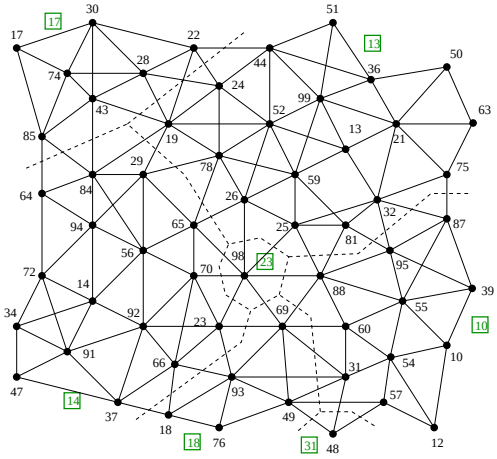
Figure 3.1: An Example Network

(Figures 3.2(e) through 3.2(l)) values, where a zone is defined to be the set of processes whose value of $minid[i]$ or $maxmin[i]$, for a given i, is the same.

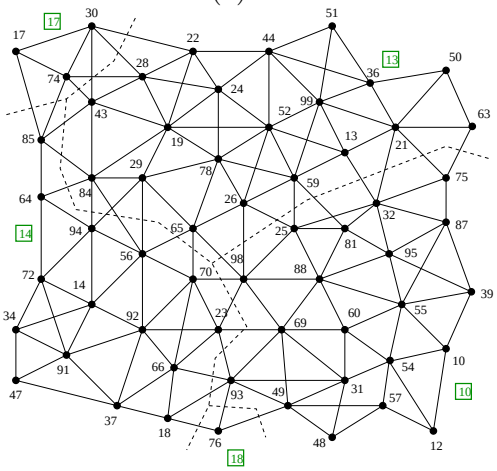
For example, in Figure 3.2(a), processes 18, 37, 66, 76, and 93 computed 18 as their $minid[1]$. In Figure 3.2(h), processes 13, 21, 32, 36, 39, 50, 63, 75, and 87 computed 13 as their $maxminid[4]$ and their final clusterhead. Note that in Figure 3.2(d), process 14, which will be chosen to be a clusterhead because it is the *minid* of some processes, is not a member of its own zone.



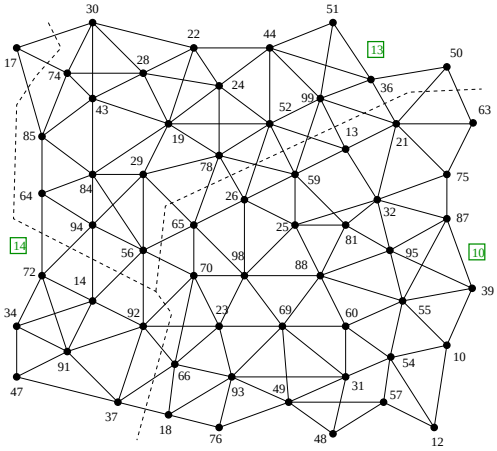
(a)



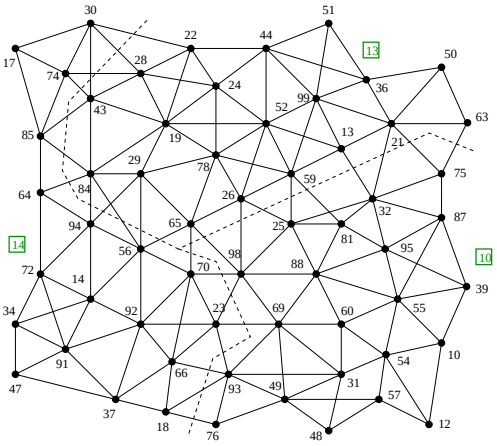
(b)



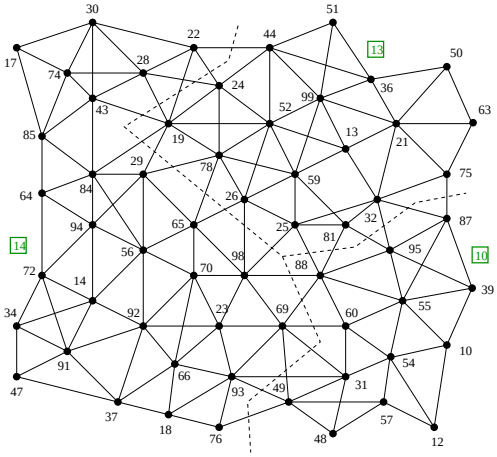
(c)



(d)



(e)



(f)

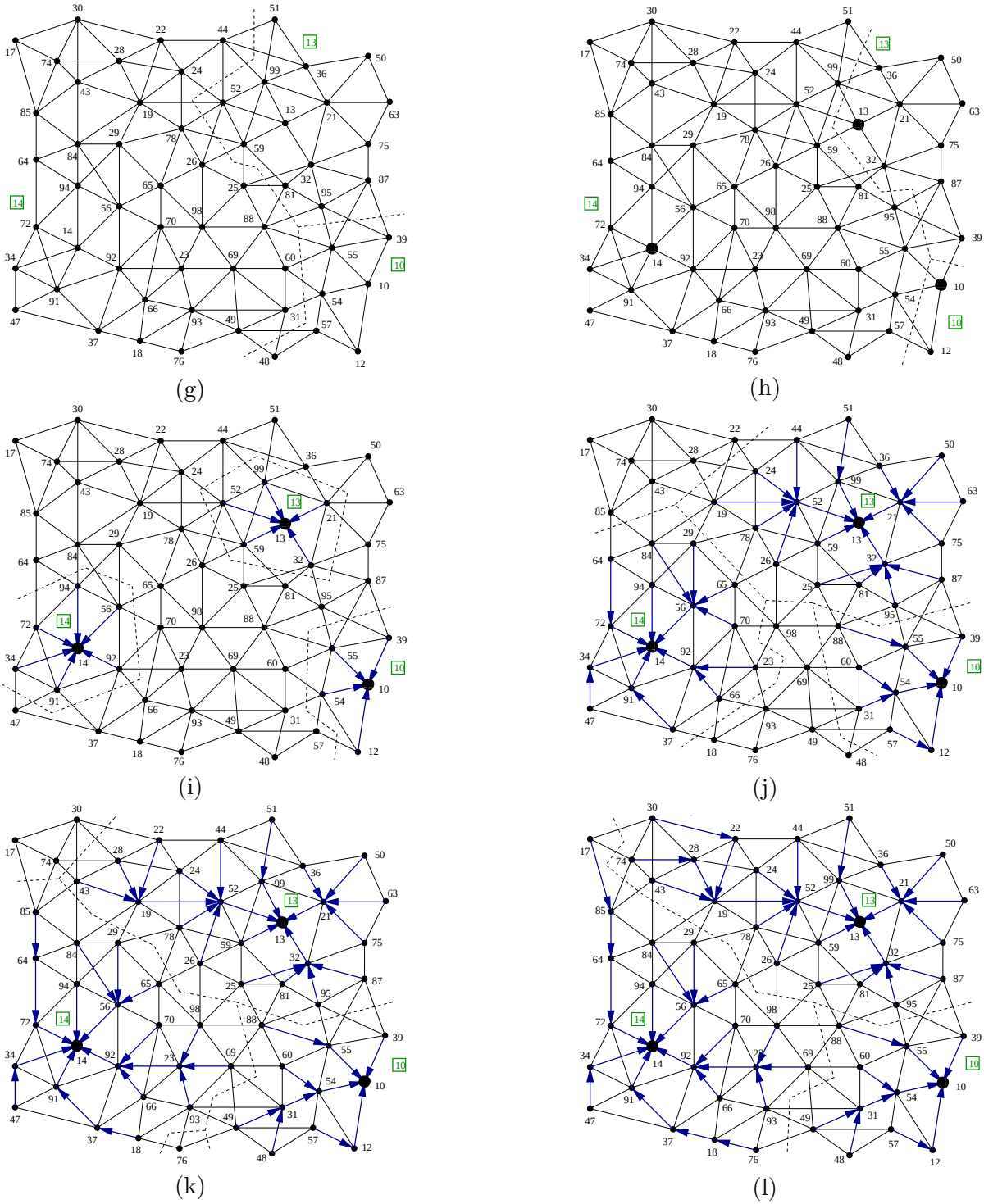


Figure 3.2: Sequence of configurations illustrating FLOOD.

3.3 Proofs for FLOOD

To aid in our proofs, we define a number of functions. Unlike the functions introduced in Section 3.1, which can be computed by a process P, these functions are defined abstractly.

$DistA(P,Q)$ = the distance, *i.e.,* number of hops, from P to Q.

$\mathcal{H}_d(P) = \{Q : DistA(P,Q) \leq d\}$.

$MinIdA(P,d) = \min\{Q.id : Q \in \mathcal{H}_d(P)\}$.

$MaxMinIdA(P,d) = \max\{MinIdA(Q,k) : Q \in \mathcal{H}_d(P)\}$.

$IsClusterheadA(P) \equiv \exists Q : MinIdA(Q,k) = P.id$.

$DistA(P) = \min\{DistA(P,Q) : IsClusterheadA(Q)\}$.

The similarity of the names of the abstract functions given above and the locally computable functions given in Section 3.1 is deliberate. For example, $MinIdA(P,d)$ is unchangeable, and is not immediately computable by P, while $MinIdF(P,d)$ is changeable, and is computable by P at any time. We shall show that, eventually, the computable value of $MinIdF(P,d)$, as well as the variable $P.minid[d]$, will be equal to $MinIdA(P,d)$.

Lemma 3.1 *$IsClusterheadA(P)$ if and only if $MaxMinIdA(P,k) = P.id$.*

Proof: One direction is easy: if $MaxMinIdA(P,k) = P.id$, then $MinIdA(Q,k) = P.id$ for some $Q \in \mathcal{H}_k(P)$, *i.e., $IsClusterheadA(P)$*. We prove the converse by contradiction. Suppose that $IsClusterheadA(P)$. Pick Q such that $MinIdA(Q,k) = P.id$. By definition, $P \in \mathcal{H}_k(Q)$, which implies that $Q \in \mathcal{H}_k(P)$.
Suppose $MaxMinIdA(P,k) = MinIdA(R,k) = S.id$. If $S.id > P.id$, then that contradicts the definition of $MinIdA(R,k)$, since $P.id$ would be a better choice. If $S.id < P.id$, that contradicts the definition of $MaxMinIdA(P,k)$, since $P.id$ would be a better choice than $S.id$. Thus, $S = P$. □

Lemma 3.2 *Let P be a process.*
(a) *If at least t rounds have elapsed, then $P.minid[d] = MinIdA(P,d)$ for all $1 \leq d \leq \min\{t,k\}$.*
(b) *After k rounds have elapsed, Action A1 is silent.*
(c) *If at least t rounds have elapsed, then $P.maxminid[d] = MaxMinIdA(P,d)$ for all $1 \leq d \leq \min\{t-k,k\}$.*
(d) *After $2k$ rounds have elapsed, Actions A1 and A2 are silent.*
(e) *If at least t rounds have elapsed, then $P.dist \geq \min\{t-2k, DistA(P)\}$.*
(f) *If at least t rounds have elapsed and $Dist(P) \leq t-2k-1$, then $P.dist = DistA(P)$.*
(g) *If at least t rounds have elapsed, and if $DistA(P) \leq t-2k-1$, then $P.leader = LeaderA(P)$ and $P.parent = ParentA(P)$.*

Proof: We prove Part (a) by induction on t. Action A1 is enabled to execute whenever its guard is true, since no action has a lower priority number.

Let $t = 1$. $MinIdF(P,1) = MinIdA(P,1)$ permanently, since both have the same definition. P is enabled to execute Action A1(1) if $P.minid[1] \neq MinIdF(P,1)$. Thus, within one round, $P.minid[1] = MinIdA(P,1)$

Suppose $t \geq 2$. If $d < t$, we are done, by the inductive hypothesis. Let $d = t$. After $t-1$ rounds, by the inductive hypothesis, $Q.\min id[t-1] = MinIdA(Q,t-1)$ for all Q, and hence $MinIdF(P,t) = MinIdA(P,t)$; and by the inductive hypothesis, all actions of priority numbers

$1 \ldots d-1$ are silent, and thus P is enabled to execute Action A1(d) if its guard is true. Within one more round, $P.minid[t] = MinIdA(P,t)$.

Part (b) follows from (a), by letting $t = k$.

We prove Part (c) by induction on t.

If $t < k+1$, the statement is vacuous. Suppose $t = k+1$. Then $d = 1$.

By (a), $Q.min[k] = MinIdA(Q,k)$ for all $Q \in \mathcal{U}_P$ after k rounds have elapsed, and thus, by definition, $MaxMinIdF(P,1) = MaxMinIdA(P,1)$.

By (b), P is enabled to execute Action A2(1) if $P.maxminid[1] \neq MaxMinIdF(P,1)$. Thus, within one more round, $P.maxminid[1] = MaxMinIdA(P,1)$

Suppose $t > k+1$. Without loss of generality, $t \leq 2k$. If $d < t-k$, we are done, by the inductive hypothesis. Let $d = t-k$. By the inductive hypothesis, $Q.maxmin[d-1] = MinIdA(Q,d-1)$ for all $Q \in \mathcal{U}_P$ after $t-1$ rounds have elapsed, and thus $MaxMinIdF(P,d) = MaxMinIdA(P,d)$.

By (b) and by the inductive hypothesis, P is enabled to execute Action A2(d) if $P.maxminid[d] \neq MaxMinIdF(P,d)$. Thus, within one more round, $P.maxminid[d] = MaxMinIdA(P,d)$.

Part (d) follows from (c), by letting $t = 2k$, and from Part (b).

We prove Part (e) by induction on t.

If $t \leq 2k$ or $DistA(P) = 0$, we are done, since $P.dist$ cannot be negative.

Suppose $t > 2k$ and $d = Dist(P) > 0$. If $d < t-2k$, we are done by the inductive hypothesis. Let $d = t-2k$. After $t-1$ rounds have elapsed, $Q.dist \geq d-1$ for all $Q \in \mathcal{N}_P$, and hence $DistF(P) \geq d$, by the inductive hypothesis. By (d), P is enabled to execute Action A3 if $P.dist \neq DistF(P)$. Thus, after one more round, $P.dist \geq d$.

We prove Part (f) by induction on t. Let $d = DistA(P)$. If $t = 2k+1$ and $d = 0$, then, by (d), P is enabled to execute Action A3 if $P.dist \neq DistF(P)$. Thus, after one more round, $P.dist = 0$.

Suppose $t > 2k+1$. If $d < t-2k-1$, we are done by the inductive hypothesis. Let $d = t-2k-1$. Pick $Q \in \mathcal{N}_P$ such that $DistA(Q) = d-1$. By the inductive hypothesis, after $t-1$ rounds, $Q.dist = d-1$, hence $DistF(P) \leq d$; and P is enabled to execute Action A3 if $P.dist \neq DistF(P)$. Thus, after one more round, $P.dist \leq d$. By Part (e), $P.dist = d$.

We prove Part (g) by induction on t. Let $d = DistA(P)$. If $t = 2k+1$ and $d = 0$, then, by (c), after $t-1$ rounds have elapsed, $LeaderF(P) = LeaderA(P) = P.id$ and $ParentF(P) = ParentA(P) = P.id$, and by (d), P is enabled to execute Action A3 if its guard is true. Thus, after one more round, $P.leader = P.parent = P.id$.

Let $t > 2k+1$. If $d < t-2k-1$, we are done by the inductive hypothesis. Let $d = t-2k-1$. Pick $Q \in \mathcal{N}_P$ such that $ParentA(P) = Q.id$. By (f), $Q.dist = d-1$ and, by the inductive hypothesis, $Q.leader = LeaderA(Q) = LeaderA(P)$ after $t-1$ rounds have elapsed. We need to show that $ParentF(P) = Q.id$ and $LeaderF(P) = Q.leader$ after $t-1$ rounds have elapsed.

If $R \in \mathcal{N}_P$, $R \neq Q$, then, by definition of $ParentA(P)$, $DistA(Q) \geq d-1$, and $R.id > Q.id$ if $DistA(R) = d-1$. By (e), if $t-1$ rounds have elapsed, $DistF(R) \geq d-1$ and $DistF(R) > d-1$ if $DistA(R) > d-1$, hence $ParentF(P) \neq R$. The statement of Action 3 consists of three parts, executed in sequence. If that action is not enabled, we are done. Otherwise, after the first part has executed, $P.dist = d$, by (f), and $ParentA(P) = Q$. After the second part has

executed, $P.parent = Q.id$ and $LeaderF(P) = Q.leader$. After the third part has executed, $P.leader = Q.leader$, and we are done. □

Theorem 3.3 FLOOD *stabilizes within* $3k + 1$ *rounds of initialization, and partitions the processes into* k*-clusters. The processes of each cluster form a BFS tree, of height at most* k*, rooted at the clusterhead.*

Proof: For each process P, the process whose ID is $MinIdA(P, k)$ is a clusterhead, and thus P is within k hops of some clusterhead. Thus, by Lemma 3.2, FLOOD is silent after $3k + 1$ rounds. By the definitions of $DistA$, $ParentA$, and $LeaderA$, and by Lemma 3.2, $\{P : P.leader = R.id\}$ is a k-cluster for any clusterhead R, and contains an internal BFS tree rooted at R defined by the *parent* pointers. □

4 A Bad Case

In this subsection, we show that, in the worst case, FLOOD picks most processes to be clusterheads, even in the special case of a planar disk graph.

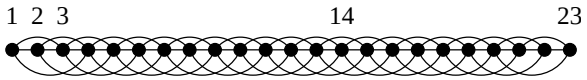


Figure 4.1: The Line Graph $\mathcal{L}_{23,3}$.

The Line Graph $\mathcal{L}_{n,m}$. For any integers $1 \leq m < n$, define $\mathcal{L}_{n,m}$ to be the network consisting of processes $P_1, \ldots P_n$, where $P_i.id = i$, and where P_i is adjacent to P_j if and only if $|i - j| \leq m$.

Figure 4.1 shows the graph $\mathcal{L}_{23,3}$. Note that $\mathcal{L}_{n,m}$ can be realized as a unit disk graph in the line (and hence the plane), by placing each P_i at the point whose coordinate is $\frac{2i}{m}$.

Lemma 4.1 *Consider the* k*-clustering chosen by* FLOOD *on* $\mathcal{L}_{n,m}$*, for any given* n*,* m*, and* k*. Then* P_i *is a clusterhead if and only if* $i \leq n - mk$*.*

Proof: $MinIdA(P_j) = \begin{cases} j - mk & \text{if } j > mk \\ 1 & \text{otherwise} \end{cases}$ □

Thus, If $mk \leq \frac{1}{2}n$, most of the processes of $\mathcal{L}_{n,m}$ will be chosen to be clusterheads by FLOOD.

5 A Lower Bound for Comparison Based Clustering Algorithms

We now show that the worst case behavior illustrated in 4 is unavoidable for any fast algorithm that uses only comparison to distinguish IDs.

We define an algorithm for the k-clustering problem to be *comparison based* if the only operator permitted on IDs is comparison. For example, the algorithm FLOOD given in this paper is comparison based. In contrast, an algorithm that can do arithmetic on an ID, such as computing $P.id \mod 2$ or extracting a single bit from an $P.id$, is not comparison based.

Theorem 5.1 *There is no comparison based deterministic distributed algorithm for the* k*-clustering problem that takes* $o(diam)$ *time, where diam is the diameter of the network, and selects fewer than half of all processes to be clusterheads. Furthermore, there is no function of* k *which is an upper bound on the competitiveness of such an algorithm.*

Proof: Let k be given. Suppose that $\mathcal{A}$ is a comparison based deterministic distributed algorithm for the k-clustering problem that takes at most $r \geq k$ rounds for any network. Pick $n = 4r + 2$, and let the network be $\mathcal{L}_{n,1}$.

We start $\mathcal{A}$ in a configuration where all processes have the same values of their variables. Consider an adversary which selects all enabled processes at every step. All behavior of a P_i during the first t rounds is determined by the initial states of the processes in the "window" around P_i of radius t, *i.e.,* $\{P_j : |i - j| \leq t\}$. Because comparison is the only operator permitted for IDs, the windows of radius r for all P_i such that $r + 1 \leq i \leq n - r - 1$ are indistinguishable to $\mathcal{A}$. Thus, either all those processes will be chosen to be clusterheads, or none will. Choosing none is impossible, since the middle processes would not be in any cluster. Thus, all the processes in that range, more than half the processes altogether, will be clusterheads.

To prove the second part, assume that $\mathcal{A}$ has competitiveness C_k. Let $n = \frac{4rC_k}{k} + 2$, and let $m = \frac{C_k}{k}$. Let the network be $\mathcal{L}_{n,C_k/k}$. Using essentially the same argument as above, we can show that $\mathcal{A}$ must choose more than half the processes to be clusterheads; but the optimal k-clustering consists of only $\frac{n}{2C_k}$ clusters. □

6 Conclusion

In this paper, we present a self-stabilizing asynchronous distributed algorithms for construction of a k-dominating set, and hence a k-clustering, for a given k, for any network with unique IDs. Our algorithm uses $O(k \log n)$ space and $O(k)$ rounds time.

We also give a lower bound tradeoff between the time complexity and competitiveness of any distributed algorithm for the k-clustering problem that uses only comparison to distinguish IDs. Any such algorithm that is C-competitive for any $C < \frac{n}{2}$ must take $\Omega(diam)$ rounds in the worst case.

References

[1] A D Amis, R Prakash, D H, and T Vuong. Max-min d-cluster formation in wireless ad hoc networks. In *IEEE INFOCOM*, pages 32–41, 2000.

[2] EW Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17:643–644, 1974.

[3] S Dolev. *Self-Stabilization.* The MIT Press, 2000.

[4] S Dolev, A Israeli, and S Moran. Uniform dynamic self-stabilizing leader election. *IEEE Transactions on Parallel and Distributed Systems*, 8(4):424–440, 1997.

[5] Y Fernandess and D Malkhi. K-clustering in wireless ad hoc networks. In *ACM Workshop on Principles of Mobile Computing POMC 2002*, pages 31–37, 2002.

[6] Shay Kutten and David Peleg. Fast distributed construction of small -dominating sets and applications. *J. Algorithms*, 28(1):40–66, 1998.

[7] Lucia Draque Penso and Valmir C. Barbosa. A distributed algorithm to find k-dominating sets. *Discrete Applied Mathematics*, 141(1-3):243–253, 2004.

[8] Vlady Ravelomanana. Distributed k-clustering algorithms for random wireless multihop networks. In *ICN (1)*, pages 109–116, 2005.

[9] M A Spohn and J J Garcia-Luna-Aceves. Bounded-distance multi-clusterhead formation in wireless ad hoc networks. *Ad Hoc Networks*, 5:504–530, 2004.