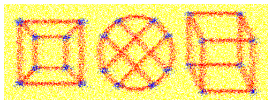


Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 19, no. 1, pp. 223–242 (2015)
DOI: 10.7155/jgaa.00354

A Distributed Algorithm for Minimum Distance-k Domination in Trees

Volker Turau Sven Köhler

Institute of Telematics, Hamburg University of Technology, Hamburg, Germany

Abstract

While efficient algorithms for finding minimal distance-k dominating sets exist, finding minimum such sets is NP-hard even for bipartite graphs. This paper presents a distributed algorithm to determine a minimum (connected) distance-k dominating set and a maximum distance-$2k$ independent set of a tree T. It terminates in $O(height(T))$ rounds and uses $O(\log k)$ space. To the best of our knowledge this is the first distributed algorithm that computes a minimum (as opposed to a minimal) distance-k dominating set for trees. The algorithm can also be applied to general graphs, albeit the distance-k dominating sets are not necessarily minimal.

Submitted:	Reviewed:	Revised:	Reviewed:	Revised:
January 2014	June 2014	July 2014	November 2014	December 2014
	Accepted:	Final:	Published:	
	March 2015	March 2015	March 2015	
	Article type:	Communicated by:		
	Regular paper	S. Whitesides		

Research funded by Deutsche Forschungsgemeinschaft (DFG), contract number TU 221/6-1
E-mail addresses: turau@tuhh.de (Volker Turau) sven.koehler@tu-harburg.de (Sven Köhler)

1 Introduction

The dominating and independent set problems are long known to be NP-complete even for substantially restricted graph classes. This paper considers generalizations of these two problems in a distributed setting. Let $G = (V, E)$ be an undirected graph and $k > 0$ an integer. A *distance-k dominating set* of G is a subset D of V such that for each $v \in V$ there exists $u \in D$ with $dist(v, u) \leq k$. While there are efficient algorithms for finding minimal distance-k dominating sets, finding minimum sets is NP-hard even for bipartite graphs. A subset $I \subseteq V$ is called *distance-k independent* if the distance between any two nodes of I is greater than k. A maximum distance-k independent set is also a minimal distance-k dominating set, but not necessarily a minimum such set. Finding a maximum distance-k independent set of a graph is NP-hard. For $k \geq 2$ this even holds for regular bipartite graphs [14]. Maximum distance-k independent sets can be computed for chordal graphs in polynomial time for odd k, whereas for even k the corresponding decision problem on this class is NP-complete [10].

The following correlation can be exploited to obtain sequential algorithms for distance-k related problems. The k-th power of G is the graph $G^k = (V, E^k)$ with $(v, w) \in E^k$ for $v \neq w$ iff the distance of v and w in G is at most k. A subset $U \subseteq V$ is a distance-k dominating set in G iff U is a dominating set in G^k. Thus, the application of an algorithm for a minimum dominating set algorithm in G^k gives a minimum distance-k dominating set for G (a similar result holds for distance-k independent sets). There are two obstacles to use this technique in distributed algorithms. Firstly, emulating the graph G^k in G comes with a high communication cost. Secondly, many structural properties of graphs are not preserved by powers, e.g., powers of trees are no longer trees. Thus, a distributed algorithm requires a different approach.

This paper presents distributed algorithms for distance-k problems in trees. In particular we present a distributed linear-time algorithm for computing minimum distance-k dominating sets of trees. For even k the algorithm computes two different distance-k dominating sets, one of them is also distance-k independent. The algorithm can be applied to general graphs, but the computed distance-k dominating set is not necessarily minimal. As a by-product a maximum distance-$2k$ independent set and a minimum connected distance-k dominating set of a tree is computed.

1.1 Related Work

Distance-k dominating sets are an important tool for dealing with the inherent lack of an infrastructure in ad hoc and wireless sensor networks. In this domain the concept is better known under the name clustering [1]. Clustering allows the formation of virtual backbones to improve the usage of scarce resources, such as bandwidth and energy. In many applications of these networks nodes need to frequently flood control messages to discover and maintain routes, which causes performance problems in terms of unnecessary traffic, energy consumption, contention, and collision. A connected k-hop dominating set can be used to dra-

matically reduce the flooding search space. Messages are forwarded along this tree and each node of the distance-k dominating set starts a restricted flooding within its cluster. This approach results in significant flooding overhead reduction for all broadcast related applications [30, 31, 28]. This demonstrates the need for distributed algorithms for computing connected distance-k dominating sets. Wu and Li give an overview of further application patterns for (connected) distance-k dominating sets in wireless sensor networks in [29]. Connected k-hop dominating sets may become disconnected due to node mobility or switch-off, which necessitates the reformation of the k-hop dominating set. To dynamically react upon such network changes self-stabilization is a widely accepted solution.

Further applications of distance-k dominating based clustering such as efficient network initialization are discussed in [15]. Penso and Barbosa report about the usage of distance-k dominating sets of small sizes in a variety of contexts, including multicast systems, the placement of servers in a computer network, the caching of replicas in database and operating systems, and message routing with sparse tables [21]. The problem of allocating and utilizing centers in communication networks can also be solved with distance-k dominating sets. Using a collection of centers offers an intermediate approach between centralized and fully distributed solutions, and provides a reasonable balance between the need for fault-tolerance and economical considerations [3].

The only distributed algorithm for computing minimum and not only minimal distance-k dominating sets is due to Wang et al. [27]. Their algorithm assumes split-star graphs, but handles only the cases $k = 1, 2$. The ideas cannot be carried over to the case $k > 2$. All other distributed algorithms compute at best minimal such sets, e.g. [21, 15, 8]. Jia et al. presented an algorithm for distance-1 dominating sets that runs in $O(\log n \log \Delta)$ rounds with high probability [12]. The size of the obtained dominating set is within $O(\log \Delta)$ of the optimal size in expectation. Schneider and Wattenhofer present a deterministic algorithm that computes in $O(\log^* n)$ rounds a connected distance-1 dominating set with constant approximation ratio for bounded-independence graphs [22]. Peleg describes a distributed algorithm to compute a distance-1 dominating set of size at most $n/2$ for trees with time complexity $O(\log^* n)$ [20].

One of the few works for $k > 1$ is due to Datta et al. [7]. They present a distributed self-stabilizing algorithm to compute a distance-k dominating set D. For unit disk graphs the size of D at most $7.2552k + O(1)$ times the minimum possible size. The algorithm presented in this paper is similar to that in [7]. Our main contribution is a proof that, on trees, the algorithm in fact computes a minimum and not just a minimal distance-k dominating set. Furthermore, we extend the algorithm to also compute a maximum distance-$2k$ independent set and a minimum connected distance-k dominating set.

Métivier et al. present a randomized distributed maximal distance-1 independent set algorithm for general graphs terminating in $O(\log n)$ rounds with probability $1 - o(n^{-1})$ [19]. The algorithm uses techniques introduced by Luby [17]. There are only a few distributed algorithms for distance-k independent sets with $k > 1$. Self-stabilizing solutions are described in [18] for arbitrary k and in [23] (resp. [26]) for $k = 2$ (resp. $k = 1$). They compute maximal and

not maximum solutions. In the first case the required number of rounds grows exponentially with n.

Using the work of Awerbuch and Sipser it is possible to transform deterministic distributed algorithms into self-stabilizing algorithms (see [16, 2]). The price to pay is among other things an increased number of messages and additional local storage depending on the run-time complexity of the algorithm. Using this transformation concept some of the above mentioned algorithms can be made self-stabilizing.

Deterministic efficient distributed approximation algorithms with a guaranteed approximation ratio for optimization problems for which no exact polynomial sequential algorithm is known are extremely rare. Efficient distributed algorithms that exactly solve such problems for restricted classes of graphs are also not very common. These statements in particular hold for the distance-k dominating and the distance-k independent set problems. To the best of our knowledge the only work that comes close to this goal is a deterministic distributed algorithm to compute a maximal distance-1 independent set of size at most n/2 for a tree with time complexity $O(\log^* n)$ [20]. Cockayne et al. have developed a sequential algorithm for minimum dominating set (i.e. $k = 1$) of a tree that runs in linear time [6]. A sequential linear algorithm for the more general problem of R-domination is due to Slater [24]. These sequential algorithms do not directly lead to distributed algorithms. This paper presents the first distributed algorithm that computes exact solutions for two notoriously difficult problems – distance-k domination and distance-$2k$ independence – for the class of trees.

2 Notation

Let $G = (V, E)$ be an undirected graph and $k > 0$. A *distance-k dominating set* of G is a subset D of V such that for each $v \in V$ there exists $u \in D$ with $dist(v, u) \leq k$[1]. Nodes in D are called *dominators*. A distance-k dominating set D is called *minimum* (resp. *minimal*) if there exists no other distance-k dominating set D' with $|D'| < |D|$ (resp. $D' \subset D$). The size of a minimum such set D is denoted by $\gamma_k(G)$. A distance-k dominating set D is called *connected* if the subgraph induced by D is connected. Denote by $Diam(G)$ the diameter of G.

A subset $I \subseteq V$ is called *distance-k independent* (a.k.a. *distance-k packing*) if the distance between any two distinct nodes of I is greater than k. It is called *maximum* (resp. *maximal*) distance-k independent if there exists no other distance-k independent set I' with $|I'| > |I|$ (resp. $I' \supset I$).

Let T be a rooted tree. The edges are oriented towards the root. If (v_1, v_2) is a directed edge then v_2 is called the successor of v_1 and v_1 a predecessor of

[1] There exists a diverging notation in the literature. The described concept is sometimes called *k-dominating* set, e.g. in [8, 15], while other authors use the latter term in the following sense. A *k-dominating* set of G is a subset S of V such that each $v \in V \backslash S$ has at least k neighbors in S, e.g. [11, 13].

v_2. The set of predecessors of v is denoted by $P(v)$. An ancestor of v is a node from where v is reachable. The distance from a node v to the root is called the *height* of v. The maximal height of a node in T is called the *height* of T.

The distributed algorithms presented in this paper use the asynchronous shared memory model, more precisely the asynchronous CONGEST model as defined in [20]. In this model, each node can communicate only with its neighbors by exchanging messages of size $O(\log n)$ via the communication links. The network can be viewed as a communication graph G where nodes represent processors and links correspond to communication registers. The restriction to messages of small size is important. In a model in which messages of unlimited size are allowed, each node can within $Diam(G)$ rounds collect the information about the entire topology and thus, solve any problem – including the one considered in this work – locally.

Each node defines a finite number of variables. A node can read and write its own variables. In addition a node can read the variables of each of its neighbors, but it cannot write these variables. Apart from this shared memory, nodes can also have some local memory. The actions of the individual nodes are not coordinated, i.e., each node can run at its own speed. Time is measured in rounds. A round is the minimal time span such that each node that wants to execute an action gets a chance to execute that action. For detailed definitions of these concepts we refer to the standard literature [25, 9, 20].

3 The Algorithm

3.1 Informal Description

Let T be a rooted tree. To provide a motivation for our distributed algorithm we first sketch a sequential algorithm to compute a minimum distance-k dominating set of T. The approach cannot be directly adopted to the distributed model, but it conveys the main idea. The algorithm incrementally builds a distance-k dominating set. In each round a yet not dominated node v of T with maximal depth is selected. Its successor at distance k is marked as a member of the distance-k dominating set. If the depth of the selected node v is less than k then the root is marked. Algorithm 1 shows the complete code of this algorithm.

Lemma 1 *The nodes marked by Algorithm 1 form a minimum distance-k dominating set of T.*

Proof: Let $v_1, \ldots, v_s$ be the selected nodes and $W = \{w_1, \ldots, w_s\}$ the corresponding dominators. Furthermore, let D be any minimum distance-k dominating set of T. Obviously, W is a distance-k dominating set of T. It suffices to show that $|D| \geq |W|$. For a set U of nodes let

$$dom_k(U) = \{v \mid dist(u, v) \leq k \text{ for some node } u \in U\}.$$

Let $W_i = dom_k(\{w_1, \ldots, w_i\})$. Choose $d_1 \in D$ such that $v_1 \in dom_k(\{d_1\})$. Since v_1 has maximal depth in T we have $dom_k(\{d_1\}) \subseteq dom_k(\{w_1\})$. Suppose we have chosen $d_1, \ldots, d_{i-1} \in D$ such that $d_j \in D \backslash \{d_1, \ldots, d_{j-1}\}$, $v_j \in$

```
foreach v in T do
    v.dominated := false, v.dominator := false;
while not all nodes of T are dominated do
    let v be a node of maximal depth in T with v.dominated = false;
    if dist(v, root) ≤ k then
        w := root;
    else
        w := successor of v with distance k;
    w.dominator := true;
    foreach u in T with dist(w, u) ≤ k do
        u.dominated := true;
```

Algorithm 1: A sequential algorithm to compute a minimum distance-k dominating set of a tree T.

$dom_k(\{d_j\})$ and $dom_k(\{d_j\}) \subseteq W_j$ for $j = 1, \ldots, i-1$. We will show that there exists $d_i \in D \backslash \{d_1, \ldots, d_{i-1}\}$ such that $w_i \in dom_k(\{d_i\})$ and $dom_k(\{d_i\}) \subseteq W_i$. Since v_i is not distance-k dominated by the nodes $w_1, \ldots, w_{i-1}$ it is not contained in $dom_k(\{d_1, \ldots, d_{i-1}\})$. Hence, there exists $d_i \in D \backslash \{d_1, \ldots, d_{i-1}\}$ such that $v_i \in dom_k(\{d_i\})$. Suppose there exits a node $u \in dom_k(\{d_i\})$ not contained in W_i. By choice of v_i, the depth of u is at most that of v_i. Node u cannot be an ancestor of w_i, because this would imply $dist(w_i, u) \leq k$. Hence, $dist(v_i, u) = dist(v_i, w_i) + dist(w_i, u) > 2k$. But then $dist(v_i, u) = dist(v_i, d_i) + dist(d_i, u) \leq 2k$ leads to a contradiction. Thus, $dom_k(\{d_i\}) \subseteq W_i$. This shows $|D| \geq |W|$. □

The above described algorithm does not directly lead to an efficient algorithm in the distributed model. There are two obstacles. First, the repeated search for a node that is not yet dominated at maximal depth and secondly, the determination of the newly dominated nodes. For the first aspect note that the same distance-k dominating set is selected, if we process the nodes of T with descending depth. In each step i we mark the successors at distance k (or the root) of all nodes at depth i that are not yet dominated by dominators. This can be realized by a bottom up labeling scheme. All selected nodes at depth i are assigned the value k and this assignment is decremented for each successor until we reach the value 0. Such nodes are dominators. Note that during a single iteration the assigned numbers are unique because they originate from nodes with the same depth. This is not the case for assignments of different rounds, this problem is resolved later.

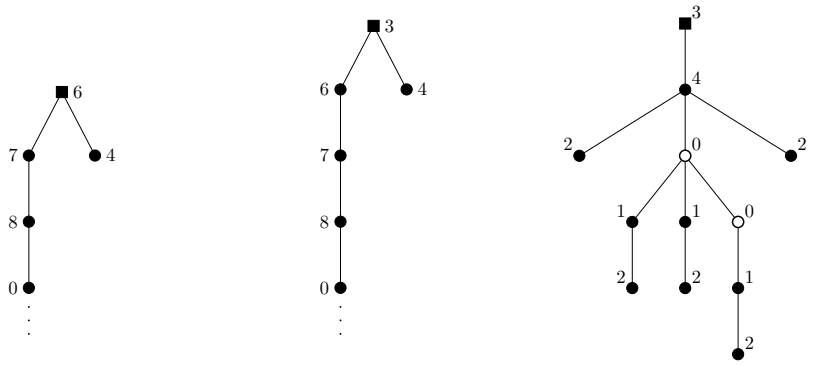
Another issue is the distributed marking of the dominated nodes. This can also be solved by assigning numbers to nodes. The successors of nodes with assignment 0 receive assignment $2k$ and this assignment is decremented for each successor until we reach the value $k + 1$. Note that all these nodes are dominated by the node with assignment 0. The main challenge is the *merging* of the different assignments into a single one. To formalize this concept the

following notation is introduced.

Definition 1 *A mapping* $\mathcal{A} : V \longrightarrow \{0, \ldots, 2k\}$ *is called a* k-assignment *of* T.

The proposed algorithm selects the same distance-k dominating set as the sequential algorithm described above. It constructs the k-assignment $\mathcal{L}$. We will show that either the set of nodes with $\mathcal{L}(w) = 0$ or those nodes together with the root form a minimum distance-k dominating set. Informally a value for $\mathcal{L}(w)$ between 1 and k indicates that the node needs a dominator within distance $\mathcal{L}(w)$. A value larger than k indicates, that there is a dominator within distance $2k + 1 - \mathcal{L}(w)$.

Informally the construction of $\mathcal{L}$ works as follows. Dominators are pushed beginning at the leaves as far as possible towards the root. The lowest dominators have distance k from the leaves, i.e., a leaf requires a dominator within distance k. Hence, $\mathcal{L}(v) = k$ for any leaf v. Successors of leaves get the value $k-1$. In general the value of a node is one less than the minimal value among its predecessors. But there are some exceptions to this rule. In case $\mathcal{L}(v) = 0$ (i.e. v is a dominator) there is no need to assign 0 to another node within distance $2k$ towards the root, but then such a node is required. Thus, a successor of a node v with $\mathcal{L}(v) = 0$ gets the value $2k$, except v has another predecessor w with $\mathcal{L}(w) = 1$. Then $\mathcal{L}(v)$ must be 0 to provide a dominator to w and its predecessors. Another exception is given if v for example has predecessors w_1, w_2 with $\mathcal{L}(w_1) = k + 3$ and $\mathcal{L}(w_2) = k$. Then w_1 provides a dominator for w_2 and v does not have to provide one. Thus, it is possible to assign $k + 2$ instead of $k - 1$ to v (see Fig. 1(a)). This is not possible in case $\mathcal{L}(w_1) = k + 2$, then it is necessary to have $\mathcal{L}(v) = k - 1$ to force v to provide a dominator for w_2 (see Fig. 1(b)).



(a) Merging of 4-assignments with rule max −1.

(b) Merging of 4-assignments with rule min −1.

(c) 2-assignment $\mathcal{L}$. Lightly colored nodes show resulting distance-2 dominating set.

Figure 1: Examples of k-assignment $\mathcal{L}$ for the cases $k = 4$ and $k = 2$.

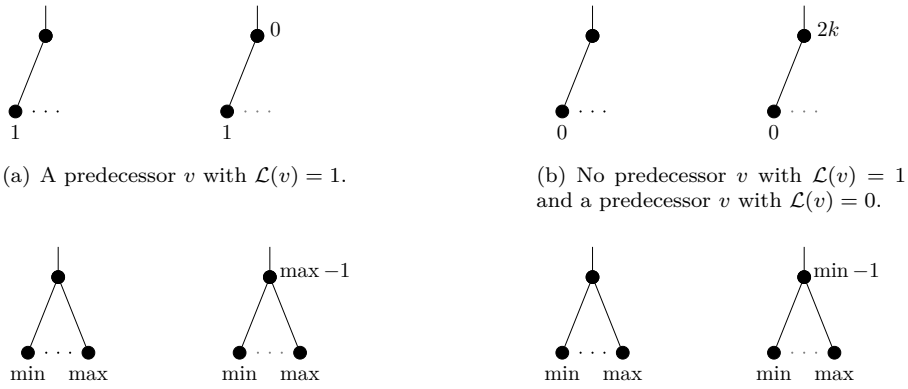
These observations lead to the following definition of $\mathcal{L}$.

Definition 2 *The k-assignment $\mathcal{L}$ is defined as follows:*

$$\mathcal{L}(v) = \begin{cases} k & \text{if } P(v) = \emptyset \text{ else} \\ 0 & \text{if } \exists w \in P(v) \text{ with } \mathcal{L}(w) = 1 \text{ else} \\ 2k & \text{if } \exists w \in P(v) \text{ with } \mathcal{L}(w) = 0 \text{ else} \\ \max -1 & \text{if } \max + \min \geq 2k+3, \\ \min -1 & \text{otherwise} \end{cases}$$

Here $\max = \max\{\mathcal{L}(v) \mid v \in P(v)\}$ *and* $\min = \min\{\mathcal{L}(v) \mid v \in P(v)\}$.

The main four cases of Def. 2 are illustrated in Fig. 2.



(a) A predecessor v with $\mathcal{L}(v) = 1$.

(b) No predecessor v with $\mathcal{L}(v) = 1$ and a predecessor v with $\mathcal{L}(v) = 0$.

(c) $\mathcal{L}(v) > 1$ for all predecessors v of w and $\max + \min \geq 2k + 3$.

(d) $\mathcal{L}(v) > 1$ for all predecessors v of w and $\max + \min < 2k + 3$.

Figure 2: Illustration of the definition of $\mathcal{L}(v)$.

The following lemma summarizes the properties of $\mathcal{L}$.

Lemma 2 *The k-assignment $\mathcal{L}$ has the following properties.*

1. *A node v with $\mathcal{L}(v) \notin \{k, 2k\}$ has a predecessor w with $\mathcal{L}(w) = \mathcal{L}(v) + 1$.*
2. *Each node v with $\mathcal{L}(v) = 2k$ has a predecessor w with $\mathcal{L}(w) = 0$.*
3. *Each node v with $\mathcal{L}(v) = k$ is either a leaf of T or has a predecessor w with $\mathcal{L}(w) = k + 1$.*

Let T be a tree. Let $D_0 = \{v \in V \mid \mathcal{L}(v) = 0\}$ and

$$D = \begin{cases} D_0 \cup \{root\} & \text{if } \mathcal{L}(root) \leq k \\ D_0 & \text{otherwise} \end{cases}$$

Fig. 1(c) shows the 2-assignment $\mathcal{L}$. The lightly colored nodes form the set D. Note that $\mathcal{L}(root) > k$. The following lemma proves an important property of D.

Lemma 3 *D is a distance-k dominating set of T.*

Proof: Let v be a node of T not contained in D. First consider the case $\mathcal{L}(v) > k$. The repeated application of Lemma 2 shows that v has an ancestor w with $\mathcal{L}(w) = 0$ and $dist(v, w) \leq 2k - \mathcal{L}(v) + 1 \leq k$. Thus, v is distance-k dominated by D. Next, consider the case $\mathcal{L}(v) \leq k$. This yields that v is not the root of T, because $v \notin D$. We will show that in this case v is distance-$\mathcal{L}(v)$ dominated. Let

$$W = \{v \in V \mid 0 < \mathcal{L}(v) \leq k \text{ and } v \text{ not distance-}\mathcal{L}(v) \text{ dominated by } D\}$$

Suppose $W \neq \emptyset$. Among all nodes in W choose v such that $\mathcal{L}(v) \leq \mathcal{L}(w)$ for all $w \in W$. Let v_1 be the successor of v in T. If $\mathcal{L}(v) = 1$ or v has a sibling w with $\mathcal{L}(w) = 1$ then $\mathcal{L}(v_1) = 0$ and thus, v is distance-k dominated by D. Hence, $\mathcal{L}(v) > 1$ and thus, $k > 1$. If $\mathcal{L}(v_1) = 2k$ then v has a sibling w with $\mathcal{L}(w) = 0$. This would imply that v is distance-k dominated by w.

Denote by max (resp. min) the maximum (resp. minimum) $\mathcal{L}$ value of a predecessor of v_1. If $\max + \min \geq 2k + 3$ then $\mathcal{L}(v_1) = \max - 1$. Note that $\min \leq \mathcal{L}(v) \leq k$ and hence $\max \geq k + 3$. Thus, $\mathcal{L}(v_1) \geq k + 2$. As shown above v_1 has an ancestor w with $\mathcal{L}(w) = 0$ and $dist(v_1, w) \leq 2k - \mathcal{L}(v_1) + 1 \leq k - 1$. This yields that v is distance-k dominated by w. Finally consider the case that $\max + \min < 2k + 3$. Then $\mathcal{L}(v_1) = min - 1 < \mathcal{L}(v)$. By choice of v node v_1 is distance-$\mathcal{L}(v_1)$ dominated by a node of D. Since $\mathcal{L}(v_1) < k$ this yields that v is distance-k dominated by a node of D. This shows that $W = \emptyset$ completing the proof. □

Definition 3 *A* fading *path of a tree T is a path of length k starting in a node v with $\mathcal{L}(v) = k$ where assigned values decrease by 1 at each node.*

Fig. 3 shows a 2-assignment with three fading paths. The lightly colored nodes form the set D. Note that $\mathcal{L}(root) \leq k$.

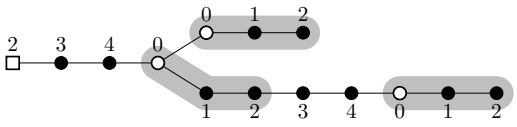


Figure 3: The three fading paths are highlighted ($\boldsymbol{k = 2}$).

Lemma 4 *The k-assignment $\mathcal{L}$ has the following properties.*

1. *Each node v with $\mathcal{L}(v) = 0$ is contained in a fading path of T.*
2. *T contains $|D_0|$ disjoint fading paths.*

Proof: The first statement follows immediately from Lemma 2. Let f be a fading path. Let T_f be the graph obtained by removing all nodes of f and edges

adjacent to nodes of f from T. Then T_f is a set of rooted trees containing $|D_0|-1$ nodes with $\mathcal{L}(w) = 0$. Furthermore, every node w of T_f with $\mathcal{L}(w) \notin \{k, 2k\}$ has a predecessor u in T_f with $\mathcal{L}(u) = \mathcal{L}(w) + 1$. Repeating this argument proves that T contains $|D_0|$ disjoint fading paths. □

Lemma 5 *Let $f = v_0, v_1, \ldots, v_k$ and $g = w_0, w_1, \ldots, w_k$ be two disjoint fading paths. Then $dist(v_j, w_j) > 2j$ for $j = 0, 1, \ldots, k$.*

Proof: Let x be the node of T with maximal depth such that v_0 and w_0 are ancestors of x. If x is neither contained in f nor in g then $dist(v_j, w_j) = dist(v_j, v_0) + dist(v_0, x) + dist(x, w_0) + dist(w_0, v_j) > 2j$. Assume $x \in f$. Let $u_0 = w_0, u_1, \ldots, u_s = x = v_i$ be the path between w_0 and f. If $s > k$ then obviously $dist(v_j, w_j) > 2j$. So let $s \leq k$. Since $\mathcal{L}(u_0) = 0$ Def. 2 yields $\mathcal{L}(u_1) \in \{0, 2k\}$. Repeated application of Def. 2 yields $\mathcal{L}(u_s) \in \{0, 1, \ldots, s-1, 2k-(s-1), \ldots, 2k\}$. Now $\mathcal{L}(u_s) = i$ implies $i \leq s-1$. Thus, $dist(w_j, v_j) = dist(w_j, w_0) + dist(w_0, v_i) + dist(v_i, v_j) \geq j + i + 1 + |i - j| > 2j$. □

Theorem 4 *D is a minimum distance-k dominating set of T.*

Proof: D is a distance-k dominating set of T by Lemma 3. If $\mathcal{L}(root) \notin \{k+1, \ldots, 2k\}$ then we can without loss of generality assume $\mathcal{L}(root) = 0$. Otherwise T can be extended to a tree T' by a path of length $\mathcal{L}(root)$ on top of *root*. Then the distance between any pair of nodes of T is the same in T and T'. This yields $D = D_0$. Let f and g be two disjoint fading paths and w_k, v_k nodes of f and g with $\mathcal{L}(w_k) = \mathcal{L}(v_k) = k$. By Lemma 5 $dist(w_k, v_k) > 2k$. Thus, w_k and v_k cannot be distance-k dominated by the same node. Lemma 4 implies $\gamma_k(T) \geq |D|$. Hence D is a minimum distance-k dominating set of T. □

Lemma 6 *$\gamma_k(T) \leq \max(1, \lfloor \frac{n}{k+1} \rfloor)$ for each tree T.*

Proof: By Theorem 4 $\gamma_k(T) = |D|$ and by Lemma 4 T contains $|D_0|$ disjoint fading paths. If $\mathcal{L}(root) \notin \{1, \ldots, k\}$ then $D = D_0$ and hence, $\lfloor \frac{n}{k+1} \rfloor) \geq |D|$. Consider the case $\mathcal{L}(root) \in \{1, \ldots, k\}$. If $D_0 = \emptyset$ then $|D| = 1$. So let $D_0 \neq \emptyset$. According to Lemma 2 there exists a path $v_k, \ldots, v_s = root$ in T with $\mathcal{L}(v_i) = i$ for $i = k, \ldots, s > 0$. None of these nodes is on a fading path. Let $v \in D_0$ be a node with minimal distance from *root* and u be the successor of v. Then obviously $\mathcal{L}(u) = 2k$. Thus, the path from *root* to u contains at least s nodes. Hence, there exist $k+1$ nodes that are not on a fading path. This yields that $n \geq |D|(k+1)$. □

3.2 Implementation

The definition of $\mathcal{L}$ leads to a sequential algorithm for computing a minimum distance-k dominating set in a bottom-up style. The tree is traversed beginning at the root in a depth first style. The purpose of this traversal is to compute $\mathcal{L}$. Whenever the depth first search finally backtracks from a node v the value of the assignment $\mathcal{L}(v)$ is computed according to Def. 2. The computation can

also be carried out in a distributed algorithm. Each node maintains a variable L. In every round each node computes $\mathcal{L}(v)$ based on the value of variable L of its predecessors using Def. 2 and assigns the result to its variable L. Thus, each node v executes the following code

while $v.L \neq \mathcal{L}(v)$ **do**
  $v.L := \mathcal{L}(v)$;

After $height(T)$ rounds all variables L have their final values. Note that the distributed algorithm is also self-stabilizing. Each leaf continuously sets its value for L to k and all other nodes continuously compute $\mathcal{L}(v)$ and assign this value to L. No initialization is needed. Since the algorithm works bottom up, there is no need to demand that a node's access to variables of a neighbor is atomic.

The following theorem summarizes the main result.

Theorem 5 *Let T be a tree and $k > 0$. A minimum distance-k dominating set of T can be computed with a distributed self-stabilizing algorithm in $height(T)$ rounds using $O(\log k)$ space per node. Furthermore, $\gamma_k(T) \leq \max(1, \lfloor \frac{n}{k+1} \rfloor)$.*

4 Distance-k Independence

It is known that for $k \geq 2$, finding a maximum distance-k independent set is NP-hard, even for regular bipartite graphs [14]. In this section it is shown that the k-assignment $\mathcal{L}$ gives rise to maximum distance-$2k$ independent sets in trees. Denote by $\mathcal{F}$ a set of $|D_0|$ disjoint fading paths of a tree T and let $I_j = \{v \in f \mid f \in \mathcal{F} \wedge \mathcal{L}(v) = j\}$ for $j = 1, \ldots, k$. If $\mathcal{L}(root) \in \{1, \ldots, k\}$ then there exists a path $v_0 = root, \ldots, v_s$ with $s = k - \mathcal{L}(v_0)$ such that $\mathcal{L}(v_i) = \mathcal{L}(v_{i+1}) - 1$ and $\mathcal{L}(v_s) = k$. This path can be regarded as a partial fading path. With this provision define $I^j_{\mathcal{L}} = I_j \cup \{v_s\}$ if $\mathcal{L}(root) \in \{1, \ldots, k\}$ and $I^j_{\mathcal{L}} = I_j$ otherwise.

Lemma 7 *In any tree T the sets $I^j_{\mathcal{L}}$ are distance-$2j$ independent for $j = 1, \ldots, k$.*

Proof: If $\mathcal{L}(root) \in \{1, \ldots, k\}$ then we can without loss of generality assume $\mathcal{L}(root) = 0$. Otherwise T can be extended to a tree T' by a path of length $\mathcal{L}(root)$ on top of *root*. Then the distance between any pair of nodes of T is the same in T and T'. The result follows from Lemma 5. □

Note that $I^j_{\mathcal{L}}$ is not necessarily a maximal distance-$2j$ independent set of T. But for $j = k$ the following result holds.

Theorem 6 *In any tree the set $I^k_{\mathcal{L}}$ is a maximum distance-$2k$ independent set.*

Proof: By Lemma 7 $I^k_{\mathcal{L}}$ is a $2k$-independent set and $|I^k_{\mathcal{L}}| = |D|$. Since any distance-$k$ dominating set must have at least of the size of any distance-$2k$ independent set, it follows from Theorem 4 that $I^k_{\mathcal{L}}$ is a maximum distance-$2k$ independent set. □

Note that Chang et al. proved that for sun-free chordal graphs (in particular for trees), distance-$2k$ independency and distance-k domination are dual problems [5]. In particular, a minimum distance-k domination set has the same cardinality as a maximum distance-$2k$ independent set for this class of graphs.

4.1 Implementation

The main task to be performed is the recognition of fading paths and the marking of the corresponding nodes on these paths. This can be integrated into a single depth-first scan of the tree. At each node v with $\mathcal{L}(v) = 0$ a new fading path begins. Thus, a sequential algorithm can compute the sets $I^j_{\mathcal{L}}$ in linear time. To identify fading paths in a distributed setting each node has a variable *fading* pointing to a predecessor. Its value is computed using the following function where, $succ(v)$ denotes the successor of node v in the tree.

$$fading(v) := \begin{cases} p \in P(v) \text{ with } p.L = 1 & \text{if } v.L = 0 \\ p \in P(v) \text{ with } p.L = v.L + 1 & \text{if } 0 < v.L < k \ \wedge \\ & \quad (succ(v).fading = v \vee v = root) \\ null & \text{otherwise} \end{cases}$$

In case there are several nodes $p \in P(v)$ that satisfy the given condition node v randomly chooses one of these nodes and retains to this choice. Thus, each node v executes the following code

while $v.L \neq \mathcal{L}(v) \vee v.fading \neq fading(v)$ **do**
  $v.L := \mathcal{L}(v)$;
  $v.fading := fading(v)$;

This code computes *fading* within k rounds. The values of variable *fading* are called *legal*, if the above equation is satisfied. The following theorem summarizes the result of this section.

Theorem 7 *In a tree T with legal values for variable fading the set $\{v \in V \mid \mathcal{L}(v) = k \wedge succ(v).fading = v\}$ is a maximum distance-$2k$ independent set. It can be computed in $height(T) + k$ rounds by a distributed self-stabilizing algorithm.*

5 Minimum Connected Distance-k Dominating Sets

A sequential algorithm for computing connected distance-k dominating sets of distance-hereditary graphs (i.e. also trees) in linear time was proposed in [4]. The algorithm does not allow an efficient distributed implementation. In the following a distributed algorithm based the k-assignment $\mathcal{L}$ is presented.

Let F be a set of $|D_0|$ disjoint fading paths of T. For each $f \in F$ denote by $f(i)$ the unique node $v \in f$ with $\mathcal{L}(v) = i$.

Lemma 8 *Let C be any connected distance-k dominating set of T and $f, g \in F$ such that $g(k)$ is not contained in the subtree rooted in $f(0)$. Then $f(0) \in C$.*

Proof: By definition of a fading path the balls of radius k around $f(k)$ and $g(k)$ are disjoint. Also any path from one ball to the other has to pass through $f(0)$. Furthermore, $dist(f(k), g(k)) > 2k$ by Lemma 5. This yields $f(0) \in C$. □

Let D_C be the set of nodes that are successors of nodes in D_0. To determine a minimum connected distance-k dominating we consider three cases. If $|D| = 1$, then D is by Theorem 4 a connected distance-k dominating set (see leftmost graph in Figure 4). Next assume $|D| > 1$. The second case covers the situation that there exists a node $u \in D_0$ such that D_C equals the set of nodes of the path from u to *root*. Let w be the second top most node of D on this path and C_w be the set of all ancestors of w that are contained in D_C including w. By Lemma 8 $C_w \subseteq C$. The second graph from the left in Figure 4 shows, that C_w can be a proper subset of C. To compute these additional nodes the following definition is needed. For each node $v \in D_C$ not contained in C_w a value $dist_l(v)$ is defined as follows.

$$v.dist_l := \begin{cases} \max(\{0\} \cup \{k+1-\mathcal{L}(u) \mid u \in P(v) \backslash D_C\}) & \text{if } v = root \\ \max(\{dist_l(succ(v))+1\} \cup \{k+1-\mathcal{L}(u) \mid u \in P(v) \backslash D_C\}) & \text{else.} \end{cases}$$

Lemma 9 *If there exist a node $u \in D_0$ such that all nodes of D_C lie on the path from u to root, then the set $\{v \in D_C \mid v.dist_l \geq k\}$ is a minimum connected distance-k dominating set of T.*

Proof: The value $v.dist_l$ for a node $v \in D_C$ is equal to the largest distance from v to a node in $T \backslash T_v$, where T_v is the subtree of T rooted in v. Thus, if $v.dist_l < k$ then the predecessor of v will distance-k dominate all nodes of $T \backslash T_v$ that are distance-k dominated by v. Furthermore, all nodes with $v.dist_l = k$ are required to distance-k dominate the furthest leaves in $T \backslash T_v$. This proves the lemma. □

The above lemma suggests to compute $v.dist_l$ for all nodes v of T. This can be done in $height(T)$ rounds. But nodes of D_C with distance larger than k to the root are contained in any minimum connected distance-k dominating as shown above, see Lemma 8. Thus, $\{v \in D_C \mid dist(root, v) \geq k \vee v.dist_l \geq k\}$ is equivalent to the description in Lemma 9. Assuming that the distances to the root are known, it suffices to compute $v.dist_l$ for nodes with $dist(root, v) < k$. This can be done in k rounds.

For the third and last case let u be the node closest to *root* that has at least two predecessors in D_c. If there is more then one node of D_0 on the path from u to *root* then let w be the second top most node of these nodes. Otherwise let $w = u$. Let C_w be the set of all ancestors of w that are contained in D_C including w. By Lemma 8 $C_w \subseteq C$. The three graphs on the right side in Figure 4 show again, that C_w can be a proper subset of C.

With the above definition of $dist_l$ the conclusion of Lemma 9 does not necessarily hold in this case. A problem arises if the the node closest to the root

with at least two predecessors in D_c has distance less than k to the root (see graph in the middle of Figure 4). This node is obviously part of any minimum connected distance-k dominating set, but it can have a value for $dist_l$ that is less then k. Therefore it is necessary to find out, whether there exists a node with this property with distance less than k to the root. This can be done within k rounds. This leads to the following lemma that holds under the assumption of the third case.

Lemma 10 *If there exists a node $u \in D_C$ with at least two predecessors in D_C and $c = \min\{dist(v, root) \mid v$ has at least two predecessors in $D_C\}$, then the set $\{v \in D_C \mid v.dist_l \geq k \vee dist(v, root) \geq \min(c, k)\}$ is a minimum connected distance-k dominating set of T.*

This lemma is proved similarly to Lemma 9. Note that the value of $\min(c, k)$ can be computed within k rounds.

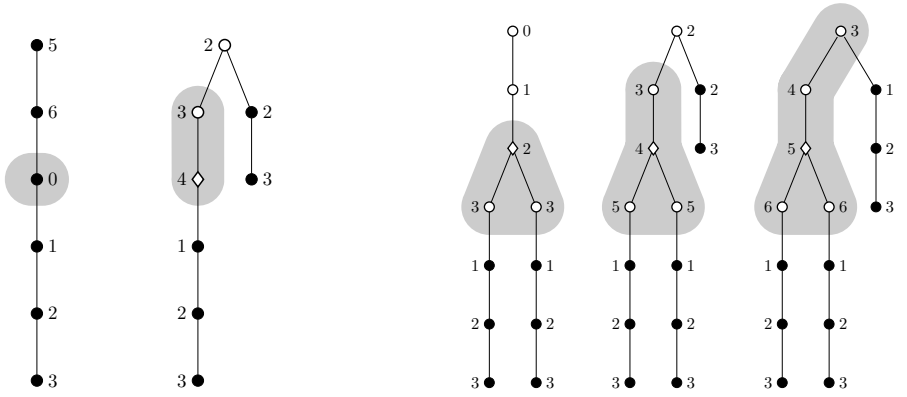


Figure 4: Connected distance-**3** dominating sets. Node $\boldsymbol{w}$ is depicted with a diamond shape. Nodes belonging to the set $\boldsymbol{D_C}$ are depicted in white, others in black. Integers next to nodes in $\boldsymbol{D_C}$ represent $\boldsymbol{dist_l}$ and $\boldsymbol{\mathcal{L}}$ for the other nodes. The nodes within the lightly colored region form the minimum connected distance-**3** dominating sets.

5.1 Implementation

To compute a minimum connected distance-k dominating set of T first the set D_0 is computed. The values of $\mathcal{L}$ are stored in a variable L as done in the last section. In the process of computing $\mathcal{L}$ the computation of $dist(root, v)$ for all nodes v can be done with no further expense. This value is stored in variable *level* of each node.

The following algorithm marks all nodes in D_C using variable *cds*. Expression $cds(v)$ evaluates to true, if there exists a predecessor w of v with $w.cds = true$. This requires additional k rounds.

```
while (v.L = k ∧ v.cds ≠ true) ∨ (v.L ≠ k ∧ v.cds ≠ cds(v)) do
    if v.L = k then
        v.cds := true;
    else
        v.cds := cds(v);
```

Next the values of $dist_l$ are computed. The algorithm uses the function $dist_l$ which can be defined on the same lines as above with the set D_C being replaced by the set of nodes with $cds = true$ and $\mathcal{L}(v)$ is replaced by $v.L$. Let $twoPred(v)$ be an expression that evaluates to true if node v has at least two predecessors with $v.cds = true$. Finally, let $minc(v)$ be a function implemented as follows.

```
if v.level ≥ k then
    return k;
else
    if twoPred(v) = true then
        return v.level;
    else
        return min{w.minc | w ∈ P(v)};
```

The purpose of this function is to compute the minimum of c and k where c is defined as in Lemma 10. Variable min is used to store the value of this function. The following code computes the values of variables $minc$ and $dist_l$.

```
while (v.cds = true ∧ v.dist_l ≠ dist_l(v)) ∨ v.minc ≠ minc(v) do
    v.dist_l := dist_l(v);
    v.minc := minc(v);
```

Note that the values of $cds, dist_l$ and $twoPred$ required to compute a minimum connected dominating set with Lemma 9 and 10 are available after $3k$ rounds. This is because the first variable is available after $2k$ rounds and the latter two variables are only required for nodes with distance at most k to the root.

The above cases can all be distinguished by the root of the tree. If $root.cds = false$ then $\{root\}$ is a minimum connected dominating set. If $root.cds = true$ but all predecessors of the root have $cds = false$ the same statement holds. If $root.cds = true$ then the set $\{v \mid v.cds = true \wedge (v.dist_l \geq k \vee v.level \geq v.minc)\}$ is a minimum connected dominating set.

The following theorem summarizes the results of this section.

Theorem 8 *There exists a self-stabilizing distributed algorithm that computes a minimum connected distance-k dominating set of a tree T in $height(T) + 3k$ rounds.*

6 General Graphs

Let T be a spanning tree of a graph G. Then obviously $\gamma_k(T) \geq \gamma_k(G)$. The following lemma suggests that the presented algorithm may also be used to compute minimal distance-k dominating sets of general graphs.

Lemma 11 *Every connected graph G has a spanning tree T with $\gamma_k(T) = \gamma_k(G)$.*

Proof: It suffices to prove that there exists a spanning tree T of G with $\gamma_k(T) \leq \gamma_k(G)$. Let D be a minimum distance-k dominating set of G. We will construct a spanning tree that contains all nodes in D. First, each node of G is connected to the closest node in D using a shortest path in G (ties are broken arbitrarily). This results in a forest F of shortest-path-trees rooted at the nodes in D containing all nodes of G. Next we add edges of G to F until it becomes a spanning tree T of G. By construction, all nodes of G are distance-k dominated in T by the nodes in D. Thus, $\gamma_k(T) \leq \gamma_k(G)$. □

Since finding a minimal distance-k dominating set is NP-hard there is only little hope to find an efficient algorithm to compute the tree T of the last Lemma without knowing D. Applying the algorithm to an arbitrary spanning tree can lead to a very poor approximation of a minimal distance-k dominating set of G. In fact it is easy to construct a graph G and a spanning tree T of G such that $\gamma_k(T)/\gamma_k(G) = \lfloor n/(k+1) \rfloor$.

In case the algorithm is applied to a spanning tree T of a graph G non-tree edges can be used to reduce the size of the resulting distance-k dominating set. So far all attempts to do so only led to heuristics without significantly improving the above stated bound. It remains an open problem whether this can lead to an approximation ratio significantly below $n/(k+1)$.

7 Conclusion

The paper presented a distributed algorithm to compute a minimum distance-k dominating set for a tree T. The algorithm stabilizes in $height(T)$ rounds requiring $O(\log k)$ storage in the distributed model. The message size is $O(\log k)$. To the best of our knowledge this is the first distributed algorithm that computes a minimum (as opposed to a minimal) distance-k dominating set for a non-trivial class of graphs. For even k the algorithm also computes a maximum distance-k independent set for a tree.

The presented distributed algorithms are self-stabilizing since they do not require any initialization and temporal errors are automatically corrected. The latter property is true since the algorithms work bottom-up with fixed values for leaf nodes. The algorithms stabilize under the unfair distributed scheduler.

Acknowledgments

The authors thank the anonymous reviewers for their constructive comments which significantly contributed to improving the quality of the publication.

References

[1] A. Abbasi and M. Younis. A survey on clustering algorithms for wireless sensor networks. *Comp. Comm.*, 30(14-15):2826 – 2841, 2007. doi:10.1016/j.comcom.2007.05.024.

[2] B. Awerbuch and M. Sipser. Dynamic networks are as fast as static networks. In *Proc. 29th Annual Symposium on Foundations of Computer Science*, SFCS '88, pages 206–219, Washington, DC, USA, 1988. IEEE Computer Society. doi:10.1109/SFCS.1988.21938.

[3] J. Bar-Ilan, G. Kortsarz, and D. Peleg. How to allocate network centers. *J. Algorithms*, 15(3):385–415, Nov. 1993. doi:10.1006/jagm.1993.1047.

[4] A. Brandstädt and F. Dragan. A linear-time algorithm for connected r-domination and steiner tree on distance-hereditary graphs. *Networks*, 31(3):177–182, 1998. doi:10.1002/(SICI)1097-0037(199805)31:3.

[5] G. Chang and G. Nemhauser. The k-domination and k-stability problems on sun-free chordal graphs. *SIAM J. Algebraic & Discrete Methods*, 5(3):332–345, 1984. doi:10.1137/0605034.

[6] E. Cockayne, S. Goodman, and S. Hedetniemi. A linear algorithm for the domination number of a tree. *Inf. Process. Lett.*, 4(2):41–44, 1975. doi:10.1016/0020-0190(75)90011-3.

[7] A. Datta, L. Larmore, S. Devismes, K. Heurtefeux, and Y. Rivierre. Competitive self-stabilizing k-clustering. In *ICDCS*, pages 476–485. IEEE, 2012. doi:10.1109/ICDCS.2012.72.

[8] A. Datta, L. Larmore, S. Devismes, K. Heurtefeux, and Y. Rivierre. Self-stabilizing small k-dominating sets. *Int. Journal of Networking and Computing*, 3(1), 2013. doi:10.1109/ICNC.2011.15.

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

[10] H. Eto, F. Guo, and E. Miyano. Distance-d independent set problems for bipartite and chordal graphs. In *Comb. Optim. & App.*, volume 7402 of *LNCS*, pages 234–244. Springer, 2012. doi:10.1007/978-3-642-31770-5_21.

[11] J. F. Fink and M. S. Jacobson. n-domination in graphs. In Y. Alavi, G. Chartrand, D. R. Lick, C. E. Wall, and L. Lesniak, editors, *Graph theory with applications to algorithms and computer science*, pages 283–300. John Wiley & Sons, Inc., New York, USA, 1985.

[12] L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. *Distrib. Comput.*, 15(4):193–205, Dec. 2002. doi:10.1017/s00446-002-0078-0.

[13] S. Kamei and H. Kakugawa. A self-stabilizing approximation algorithm for the distributed minimum k-domination. *IEICE Trans.*, 88-A(5):1109–1116, 2005. doi:10.1093/ietfec/e88-a.5.1109.

[14] M. Kong and Y. Zhao. Computing k-independent sets for regular bipartite graphs. *Congressus Numerantium*, 143:65–80, 2000.

[15] S. Kutten and D. Peleg. Fast distributed construction of small k-dominating sets and applications. *J. Algorithms*, 28(1):40–66, 1998. doi:10.1006/jagm.1998.0929.

[16] C. Lenzen, J. Suomela, and R. Wattenhofer. Local algorithms: Self-stabilization on speed. In *Proc. 11th Symposium on Stabilization, Safety, and Security of Distributed Systems, Lyon, France*, SSS '09, pages 17–34, 2009. doi:10.1007/978-3-642-05118-0_2.

[17] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

[18] F. Manne and M. Mjelde. A memory efficient self-stabilizing algorithm for maximal k-packing. In *Proc. Int. Conf. Stab. Safety, & Sec.*, pages 428–439. Springer, 2006. doi:10.1007/978-3-540-49823-0_30.

[19] Y. Métivier, J. Robson, N. Saheb-Djahromi, and A. Zemmari. An optimal bit complexity randomized distributed mis algorithm. *Distributed Computing*, 23(5-6):331–340, 2011. doi:10.1007/s00446-010-0121-5.

[20] D. Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, Philadelphia, USA, 2000. doi:10.1137/1.9780898719772.

[21] L. Penso and V. Barbosa. A distributed algorithm to find k-dominating sets. *Discrete Appl. Math.*, 141(1-3):243–253, May 2004. doi:10.1016/S0166-218X(03)00368-8.

[22] J. Schneider and R. Wattenhofer. An optimal maximal independent set algorithm for bounded-independence graphs. *Distrib. Comput.*, 22(5-6):349–361, 2010. doi:10.1007/s00446-010-0097-1.

[23] Z. Shi. A self-stabilizing algorithm to maximal 2-packing with improved complexity. *Inf. Process. Lett.*, 112(13):525–531, July 2012. doi:10.1016/j.ipl.2012.03.018.

[24] P. Slater. R-domination in graphs. *J. ACM*, 23(3):446–450, July 1976. doi:10.1145/321958.321964.

[25] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2004.

[26] V. Turau. Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler. *Inf. Process. Lett.*, 103:88–93, 2007. doi:10.1016/j.ipl.2007.02.013.

[27] F. Wang, J. Chang, Y. Wang, and S. Huang. Distributed algorithms for finding the unique minimum distance dominating set in directed split-stars. *J. Parallel Distrib. Comput.*, 63(4):481–487, Apr. 2003. doi:10.1016/S0743-7315(03)00040-6.

[28] J. Wang, Y. Yonamine, E. Kodama, and T. Takata. A distributed approach to constructing k-hop connected dominating set in ad hoc networks. In *Proc.: International Conference on Parallel and Distributed Systems (ICPADS)*, pages 357–364, Dec 2013. doi:10.1109/ICPADS.2013.57.

[29] Y. Wu and Y. Li. Connected dominating sets. *Handbook of Ad Hoc and Sensor Wireless Networks: Architectures, Algorithms and Protocols*, pages 19–39, 2009.

[30] H.-Y. Yang, C.-H. Lin, and M.-J. Tsai. Distributed algorithm for efficient construction and maintenance of connected k-hop dominating sets in mobile ad hoc networks. *Mobile Computing, IEEE Transactions on*, 7(4):444–457, April 2008. doi:10.1109/TMC.2007.70736.

[31] C. Zheng, L. Yin, and S. Sun. Construction of d-hop connected dominating sets in wireless sensor networks. *Procedia Engineering*, 15(0):3416 – 3420, 2011. doi:10.1016/j.proeng.2011.08.640.