A synchronous self-stabilizing algorithm for the minimal generalized domination in arbitrary networks

Hisaki Kobayashi† Hirotsugu Kakugawa† Toshimitsu Masuzawa†

Abstract. *A self-stabilizing system is guaranteed to eventually reach and stay at a legitimate configuration regardless of the initial configuration. In this paper, we propose a generalization of the classical k-redundant dominating set problem, and propose a self-stabilizing algorithm for finding a minimal generalized dominating set in an arbitrary network under the synchronous daemon. The classical k-redundant dominating set in a distributed system is a set of nodes such that each node is contained in the set or has k neighbors in the set. On the other hand, in our generalized dominating set, each node i is given its domination wish set* $C_i = \{W_1^i, W_2^i, \cdots : W_x^i \subseteq N_i\}$*, where* N_i *is a set of neighbors of node i, and each node i not in the dominating set has some* $W_x^i \in C_i$ *such that each member in* W_x^i *is in the dominating set. We show that the worst case stabilization time of the proposed algorithm is* $O(n)$ *rounds, where n is the number of nodes.*

1. Introduction

A *self-stabilizing* system is guaranteed to eventually reach and stay at a legitimate configuration regardless of the initial configuration [1]. This enables a distributed system to be adaptive to transient faults and topology changes in a network. Two important requirements to a self-stabilizing algorithm are *closure* and *convergence* properties. The closure property ensures that once a system reaches a legitimate configuration, it stays at legitimate configurations forever unless new transient faults or topology changes occur. The convergence property ensures that regardless of the initial configuration, the system reaches a legitimate configuration in a finite time.

A *dominating set* in a distributed system is a set of nodes such that each node is contained in the set or has at least one neighbor in the set. A *k-redundant dominating set* [4] is a set of nodes such that each node is contained in the set or has at least k neighbors in the set. We call members of a dominating set *dominators* and the remainder *dominatees*. A *minimal dominating set* is useful for clustering and routing in an ad hoc wireless network. A dominating set (resp. k-redundant dominating set) is minimal if and only if no proper subset of the set are a dominating set (resp. k-redundant dominating set). In these definitions, each node uniformly requires the same amount of domination, that is, each dominatee has at least one or k dominators in the neighbor respectively. So, these definition cannot give each node a different amount of domination. In this paper, as a further generalization of these problems, we propose the *generalized dominating set problem* in which domination requirements may not be uniform by nodes. For example, domination requirements can be decided by the network performance, the node performance, the network topology, the degree of a node and so on. In addition, a generalized dominating set can also express a weighted version of the k-redundant dominating set: given a network where each node is assigned a positive weight, each node is required to be a dominator or to be dominated by neighboring dominators whose total weight is k or more.

Contribution of this paper: The contribution of this paper is twofold. First, we introduce the generalized dominating set problem. Second, we propose a self-stabilizing algorithm for finding a minimal generalized dominating set in an arbitrary network under the synchronous daemon. In this paper, we assume the execution model where all nodes execute actions simultaneously in a lock-step fashion in each round (the synchronous daemon), and the communication model where each node can directly read local variables of neighbors (the state-reading model). These models are commonly used in literature of self-stabilization. Our algorithm repeats a sequence of four phases, and all nodes must execute an identical phase at each round. To realize the synchronization of the four phases, the self-stabilizing phase-clock synchronization algorithm [11] is utilized. The convergence time of our algorithm is $O(n)$ rounds, where n is the number of nodes.

Related works: N. Guellati and H. Kheddouci [5] surveyed self-stabilizing algorithms for finding a minimal dominating set and a minimal k-redundant dominating set under various kinds of daemons in various network topologies. The *minimum* dominating set problem is NP-hard, so several self-stabilizing algorithms [8, 15, 12, 13, 14] for the *minimal* dominating set (MDS) problem have been proposed. The first research of self-stabilizing algorithms for the minimal k-reduntant dominating set (MKDS) problem has been developed by Kamei and Kakugawa [6] which assumes a tree network under the central and the distributed daemons, and the convergence

† Graduate School of Information Science and Technology, Osaka University, 1-5 Yamadaoka, Suita, Osaka, 565-0871, Japan

times of their algorithms are both $O(n^2)$ steps. Huang et al. [9, 10] presented two self-stabilizing algorithms for the minimal 2-redundant dominating set (M2DS) problem in an arbitrary network. Recently, Wang et al. [2, 3] proposed self-stabilizing algorithms for the MKDS problem, assuming the central and the distributed daemons, both of which stabilize in $O(n^2)$ steps. The results are summarized in Table 1. Note that MGDS in the table denotes a minimal generalized dominating set proposed in this paper.

Table 1: Self-stabilizing algorithms for various dominating set problems

Reference	Problem	Topology	Daemon	Convergence time
Hedetniemi et al. [8]	MDS	Arbitrary	Central	$(2n+1)n$ steps
Xu et al. [15]	MDS	Arbitrary	Synchronous	$4n$ rounds
Turau [12]	MDS	Arbitrary	Distributed	$9n$ steps
Goddard et al. [13]	MDS	Arbitrary	Distributed	$5n$ steps
Chiu et al. [14]	MDS	Arbitrary	Distributed	$4n$ steps
Huang et al. [9]	M2DS	Arbitrary	Central	$O(n)$ steps
Huang et al. [10]	M2DS	Arbitrary	Distributed	not mentioned
Kamei and Kakugawa [6]	MKDS	Tree	Central	$O(n^2)$ steps
Kamei and Kakugawa [6]	MKDS	Tree	Distributed	$O(n^2)$ steps
Kamei and Kakugawa [7]	MKDS	Arbitrary	Synchronous	$O(n)$ steps
Wang et al. [2]	MKDS	Arbitrary	Central	$O(n^2)$ steps
Wang et al. [3]	MKDS	Arbitrary	Distributed	$O(n^2)$ steps
This paper	MGDS	Arbitrary	Synchronous	$O(n)$ rounds

Organization of this paper: The rest of the paper is organized as follows. Section 2. presents formal definitions of the system model and the generalized dominating set (MGDS) problem. Section 3. presents our algorithm for the MGDS problem under the synchronous daemon in an arbitrary network topology. Section 4. gives concluding remarks.

2. Preliminaries

2.1 System model

A *distributed system* is modeled by an undirected graph $G = (V, E)$, where $V = \{0, 1, 2, \cdots, n-1\}$ is a set of n nodes and E is a set of m bidirectional communication links. Each node $i \in V$ has a unique identifier denoted by ID_i which is a nonnegative integer value. With abuse of notation, we use i to denote ID_i when it is clear from context. N_i denotes a set of nodes to which node i is adjacent, called *neighbors*. As a communication model, we assume each node can read local states (or variables) of neighbors without delay. This model is called the *state-reading model*. Each node can update its own local state only, but each node can read local states of neighbors. A *configuration* of a distributed system G is specified by an n-tuple $\gamma = (s_0, s_1, \cdots, s_{n-1})$, where s_i stands for the state of node i $(0 \leq i \leq n-1)$. Let Γ be a set of all possible configurations. An atomic step of each node i consists of the following two steps: (1) read local states of all neighbors and (2) update its local state depending on its current state and the states read from its neighbors. In this paper, we assume the *synchronous daemon* for node execution such that all nodes execute atomic steps simultaneously in a lock-step fashion, and computations progress in rounds: in each round, every node executes an atomic step. Notice that each node reads the neighbors' states that are the ones at the beginning of the current round and updates its own state.

2.2 Self-Stabilization

When the configuration changes from γ to γ' $(\neq \Gamma)$, the transition is denoted by $\gamma \mapsto \gamma'$. For any configuration γ_0, an *execution* Π starting from γ_0 is a maximal (possibly infinite) sequence of configurations $\Pi = \gamma_0, \gamma_1, \cdots$ satisfying $\gamma_t \mapsto \gamma_{t+1}$ for each $t \geq 0$.

Definition 1. *Let Γ be the set of all possible configurations. A distributed system is* self-stabilizing *with respect to $\Lambda \subseteq \Gamma$ if and only if the following two conditions are satisfied.*

- Convergence: *Starting from an arbitrary configuration, a configuration eventually becomes one in Λ*
- Closure: *For any configuration $\gamma \in \Lambda$, any configuration γ' such that $\gamma \mapsto \gamma'$ is also in Λ.*

□

Each $\gamma \in \Lambda$ is called a *legitimate* configuration.

2.3 The Generalized Dominating Set Problem

A classical dominating set is formally defined as follows. Let $G = (V, E)$ be a distributed system.

Definition 2. *A* dominating set *D of G is a subset of V such that for each node $i \in V$, i is in D or $|N_i \cap D| \geq 1$ (or there exists j such that $j \in N_i \cap D$).* □

Definition 3. *[4] A* k-redundant dominating set *D of G is a subset of V such that for each node $i \in V$, i is in D or $|N_i \cap D| \geq k$.* □

Definition 4. *A dominating set (resp. a k-redundant dominating set) D of G is* minimal *if no proper subset of D is a dominating set (resp. a k-redundant dominating set) of G.* □

The 1-redundant dominating set problem is equivalent to the dominating set problem. Hence, the k-redundant dominating set problem is a generalization of the dominating set problem. The generalized dominating set introduced in this paper is defined as follows.

Definition 5. *Let $C_i = \{W_1^i, W_2^i, \cdots, W_{c(i)}^i\}$ for each node i $(0 \leq i \leq n-1)$ where $W_x^i \subseteq N_i$ $(1 \leq x \leq c(i))$, and let $C = (C_0, C_1, \cdots, C_{n-1})$. A* generalized dominating set *D of G with respect to C is a subset of V such that for each node i, i is in D or there exists $W_x^i \in C_i$ such that $W_x^i \subseteq D$. We call C_i a domination wish set of node i, and C a domination wish list.* □

Definition 6. *A generalized dominating set D of G is* minimal *if no proper subset of D is a generalized dominating set of G.* □

A generalized dominating set is a further generalization of a k-redundant dominating set. Besides, we define the minimal generalized dominating set problem in a distributed system.

Definition 7. *Let $G = (V, E)$ be an undirected graph modeling a distributed system. The* distributed minimal generalized dominating set problem *is defined as follows.*

Input of node i **:** *A domination wish set C_i.*

Output of node i **:** *A status $d_i = true$ or $d_i = false$.*

Condition : *A node set $\{i \in V : d_i = true\}$ is a minimal generalized dominating set of G with respect to $C = \{C_0, C_1, \cdots, C_{n-1}\}$.*

□

3. The Proposed Algorithm

3.1 Variables

In this section, we propose a self-stabilizing algorithm for the distributed minimal generalized dominating set problem. In our algorithm, each node i uses two constants, one external variable (controlled by external activity), two macro symbols and four shared variables. The constants are described as follows.

- **set of nodes** $N_i \subseteq V$: A set of neighbors of node i.
- **domination wish set** $C_i = \{W_1^i, W_2^i, \cdots : W_x^i \subseteq N_i\}$

The external variable is described as follows.

- **int** $PhaseClock_i \in \{1, 2, 3, 4\}$: We assume that external activity makes this variable increase by one (in the circular order) at each round as 1,2,3,4,1,2,3,4,1,2,$\cdots$ and take the same value in all nodes at each round. Our algorithm implicitly executes the self-stabilizing algorithm [11] for a phase clock synchronization simultaneously to maintain $PhaseClock_i$. For simplicity, in our algorithm, we omit the description of the phase clock synchronization algorithm.

Besides, we use macro symbols as follows.

- **set of nodes** $D_i = \{j \in N_i : d_j = true\}$: A set of neighboring dominators of node i. Consequently, a set $N_i - D_i$ means a set of neighboring dominatees of node i.
- $C_i' = \{W_x^i \in C_i : W_x^i \subseteq D_i\}$: A subset of C_i such that for each $W_x^i \in C_i'$, each node in W_x^i is a dominator. A dominatee i is *dominated* when $C_i' \neq \emptyset$.

The shared variables are described as follows.

- **boolean** d_i : This variable is *true* (resp. *false*) if node i is a *dominator* (resp. *dominatee*). We call this variable *status*. Note that the meanings of the *status* and *state* are different in this paper; the state means the set of the variables of node i.
- **boolean** $Permission_j^i$: This variable is used by node i to give a neighboring dominator $j(\in D_i)$ permission to become a dominatee. $Permission_j^i = true$ means that a dominatee i is dominated by the other set of dominators ($\in C_i'$) even if $j \in D_i$ turns to be a dominatee. In other case, $Permission_j^i$ is *false*.
- **boolean** $ChangeFlag_i$: Node i sets this variable *true* if node i intends to change its *status* from a dominator to a dominatee or from a dominatee to a dominator.
- **node name** $Pointer_i$: This variable is assigned one node $j \in N_i \cup \{i\}$ to approve j's status change. Node j can change its status if $Pointer_l$ points to j for each node $l \in N_j \cup \{j\}$.

3.2 Algorithm Outline

Let us explain the idea of the proposed algorithm. The main feature of our algorithm is that once a dominator i turns to be a dominatee, node i never changes its status afterwards, that is, node i is dominated by at least one set of dominators in C_i' afterwards. Intuition of the status change rules of each node i is described as follows.

· **Rule 1 dominatee → dominator :** A dominatee i ($i.e., d_i = false$) turns to be a dominator ($i.e., d_i = true$) if it is not dominated, that is, $C_i' = \emptyset$.

· **Rule 2 dominator → dominatee :** A dominator i turns to be a dominatee if it is dominated and each neighboring dominatee $j(\in N_i - D_i)$ is also dominated even if node i turns to be a dominatee.

This idea for the algorithm seems intuitively correct; Rule 1 makes a set dominating, and Rule 2 makes a set minimal. However, its straightforward implementation does not work correctly under the synchronous daemon which is assumed in this paper. Let us observe three nodes, say i, j and k in the network such that nodes j and k are neighbors of node i, but nodes j and k are not neighbors each other, that is, node i is in the middle of nodes j and k. Suppose that node i is a dominatee with $C_i = \{\{j\}, \{k\}\}$, and nodes j and k are dominators. By Rule 2, nodes j and k simultaneously become dominatees if each of them has at least one set of dominators in C_j and C_k respectively. Then, node i has no set of dominators in C_i, and node i is still a dominatee; node i is not dominated.

To avoid such a scenario, we disallow the simultaneous status changes of nodes j and k in the above setting. Generally speaking, we avoid violation of domination by disallowing simultaneous status changes of two nodes within distance two (*e.g*, nodes j and k in the above example). The idea for such a control is described below.

- Each node i reads the status from each of its neighbors. Node i can now detect whether the condition of Rule 1 is satisfied. Concerning Rule 2 at each neighboring dominator j, node i can detect whether it is still dominated even if node j turns to be a dominatee. If so, node i notifies node j of permission to become a dominatee.
- According to the permissions, each dominator can know whether or not the condition of Rule 2 is satisfied. When node i satisfies the condition of Rule 1 or Rule 2, it notifies its neighbors that it intends to change its status by setting $ChangeFlag_i := true$.
- To disallow nodes within distance two to simultaneously change their statuses, we use the pointer $Pointer_i$; node i sets $Pointer_i := j$ where $j \in N_i \cup \{i\}$ is the node with the smallest ID among $\{h \in N_i \cup \{i\} : ChangeFrag_h = true\}$. Node i sets $Pointer_i := null$ if no node h in $N_i \cup \{i\}$ satisfies $ChangeFrag_h = true$.
- After the pointer assignment, node i changes its status if node i is pointed by all the neighbors and itself.

By this, we prevent the simultaneous status changes by nodes within distance two. The algorithm repeats the sequence of the four phases. We explain the action of node i in each phase as follows.

- **Phase 1:** Each node i updates $Permission_j^i$ for each neighbor j. Node i sets $Permission_j^i := true$ if node i is a dominatee and $\{s \in C_i' : j \notin s\} \neq \emptyset$ holds. In other case, $Permission_j^i = false$. This variable is used in Phase 2.
- **Phase 2:** Each node i updates $ChangeFlag_i$. Node i sets $ChangeFlag_i = true$ if the condition of Rule 1 or Rule 2 (mentioned above) is satisfied, and $ChangeFlag_i = false$ otherwise. This variable is used in Phase 3 and 4.

- **Phase 3:** Each node i updates its $Pointer_i$. Node i sets $Pointer_i$ to one node $j \in N_i \cup \{i\}$ with the smallest ID among $\{h \in N_i \cup \{i\} : ChangeFlag_h = true\}$. Node i sets $Pointer_i := null$ if there exists no neighbor j such that $ChangeFlag_j$ is *true*. This variable is used in Phase 4.
- **Phase 4:** Each node i changes its status ($d_i := \neg d_i$) if the following two conditions are satisfied.
  1. Node i intends to change its status, that is, $ChangeFlag_i = true$.
  2. Node i is pointed by each neighbor j and itself, that is, $\forall j \in N_i \cup \{i\} : Pointer_j = i$.

Because of space limitations, we omit proof of correctness.

Theorem 1. *The algorithm is a self-stabilizing algorithm for the minimal generalized dominating set problem with $O(n)$ convergence time under the synchronous daemon.*

4. Conclusion

In this paper, we presented the new generalization of a dominating set which generalizes the classical dominating set and the classical k-redundant dominating set. The generalized dominating set is a proper generalization since more general domination can be realized than the classical dominating sets. For example, each node can designate the nodes it wants to be dominated by them, which is impossible for the classical dominating set and the k-redundant dominating set. In addition, we proposed a self-stabilizing algorithm for finding a minimal generalized dominating set under the synchronous daemon. The convergence time of our algorithm is $O(n)$, where n is the number of nodes. The algorithm works even under the distributed daemon when we apply the self-stabilizing techniques for simulating the synchronous daemon under the distributed daemon.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers JP26280022, JP16K00018, JP17K19977.

References

[1] E. W. Dijkstra: Self-stabilizing systems in spite of distributed control, *Communications of the ACM 17(11), pp.643–644* (1974).

[2] G. Wang, H. Wang, X. Tao, J. Zhang: A self-stabilizing algorithm for finding a minimal k-dominating set in general networks, *In Proceedings of the International Conference on Data and Knowledge Engineering, pp.74–85* (2012).

[3] G. Wang, H. Wang, X. Tao, J. Zhang and J. Zhang: Minimising k-dominating set in arbitrary network graphs, *In Proceedings of the 9th International Conference on Advanced Data Mining and Applications, pp.120–132* (2013).

[4] J. F. Fink and M. S. Jacobson: N-domination in graphs, *John Wiley and Sons* (1985).

[5] N. Guellati and H. Kheddouci: A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs, *Journal of Parallel and Distributed Computing 70(4), pp.406–415* (2010).

[6] S. Kamei and H. Kakugawa: A self-stabilizing algorithm for the distributed minimal k-redundant dominating set problem in tree network, *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies(PDCAT), pp.720–724* (2003).

[7] S. Kamei and H. Kakugawa: A self-stabilizing approximation algorithm for the distributed minimum k-domination, *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences E88-A (5), pp.1109–1116* (2005).

[8] S. M. Hedetniemi, S. T. Hedetniemi, D. P. Jacobs and P. K. Srimani: Self-stabilizing algorithms for minimal dominating sets and maximal independent sets, *Computers & Mathematics with Applications, pp.805–811* (2003).

[9] T. C. Huang, C. Y. Chen and C. P. Wang: A linear-time self-stabilizing algorithm for the minimal 2-dominating set problem in general networks, *Journal of Information Science and Engineering 24(1), pp.175–187* (2008).

[10] T. C. Huang, J. C. Lin, C. Y. Chen and C. P. Wang: A self-stabilizing algorithm for finding a minimal 2-dominating set assuming the distributed demon model, *Computers and Mathematics with Applications 54(3), pp.350–356* (2007).

[11] T. Herman and S. Ghosh: Stabilizing phase-clocks, *Information Proceeding Letter 5(6), pp.259–265* (1995).

[12] V. Turau: Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler, *Information Processing Letters 103(3), pp.88–93* (2007).

[13] W. Goddard, S.T. Hedetniemi, D.P. Jacobs, P.K. Srimani and Z. Xu: Self-stabilizing graph protocols, *Parallel Processing Letters 18(1), pp.189–199* (2008).

[14] W. Y. Chiu, C. Chen and S. Y. Tsai: A $4n$-move self-stabilizing algorithm for the minimal dominating set problem using an unfair distributed daemon, *Information Proceeding Letter 114(5), pp.515–518* (2014).

[15] Z. Xu, S. T. Hedetniemi, W. Goddard and P. K. Srimani: A synchronous selfstabilizing minimal domination protocol in an arbitrary network graph, *Proceedings of the Fifth International Workshop on Distributed Computing, pp.26–32* (2003).