A Self-Stabilizing Minimal Dominating Set Algorithm with Safe Convergence

Hirotsugu Kakugawa and Toshimitsu Masuzawa

Department of Computer Science
Graduate School of Information Science and Technology
Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, JAPAN
{kakugawa, masuzawa}@ist.osaka-u.ac.jp

Abstract

A self-stabilizing distributed system is a fault-tolerant distributed system that tolerates any kind and any finite number of transient faults, such as message loss and memory corruption. In this paper, we formulate a concept of safe convergence in the framework of self-stabilization. An ordinary self-stabilizing algorithm has no safety guarantee while it is in converging from any initial configuration. The safe convergence property guarantees that a system quickly converges to a safe configuration, and then, it gracefully moves to an optimal configuration without breaking safety. Then, we propose a minimal independent dominating set algorithm with safe convergence property. Especially, the proposed algorithm computes the lexicographically first minimal independent dominating set according to the process identifier as a priority. The priority scheme can be arbitrarily changed such as stability, battery power and/or computation power of node.

1. Introduction

Self-stabilization is a theoretical framework of non-masking fault-tolerant distributed algorithms proposed by Dijkstra [2, 4]. Self-stabilizing algorithms can start execution from arbitrary (illegitimate) configuration and eventually configuration becomes legitimate. By this property, a self-stabilizing system tolerates any kind and any finite number of transient faults, such as message loss, memory corruption, and topology change.

By definition, a self-stabilizing system is guaranteed to converge to a legitimate configuration, as long as no transient faults occur enough long time. Although self-stabilization property works quite well for networks in which frequency of transient faults is low (compared to the time for convergence to legitimate configuration), it may not be suitable for dynamic networks, such as mobile ad hoc networks, in which transient faults and topology changes frequently occur.

1.1. Contribution of this paper

In this paper, we cope with a problem of safety guarantee during converging period by extending of self-stabilization, named *safe convergence*. Such a concept implicitly appears in several papers in the past, and we formulate the concept formally in this paper. A safely converging self-stabilizing system defines two classes of legitimate configurations. One is a set of configurations in which property or service is *feasible*, i.e., minimum quality of service is guaranteed. The other is a set of configurations in which property or service is *optimal*. We consider configuration in both classes satisfy property or service (a safety property) of a system.

When faults occur, it is better to converge to a feasible legitimate configuration as soon as possible. If no fault occurs for enough period of time, it is better to converge to an optimal legitimate configuration to provide the best quality of service. The safety convergence property requires that the system should not break a safety property while a system is moving from a feasible configuration to an optimal configuration.

Then, we consider the problem of minimal dominating set (MDS) with safe convergence. We assume, in this paper, that all processes are executed in parallel in each step for execution model (the synchronous model), and each process can read local variables of neighbor

1-4244-0054-6/06/$20.00

processes for communication model (the state-reading model). These models are adopted often in literature of self-stabilization.

Specifically, our algorithm computes a minimal dominating set in which no two dominators neighbor each other (i.e., a minimal independent dominating set, MIDS). In addition, our algorithm computes the lexicographically first MIDS according to the process identifier as a priority.

By our algorithm, configuration quickly moves to a feasible one in which a safety property (in our current setting, "dominating set is computed") is satisfied. Then, as long as no transient fault occur, a configuration eventually becomes an optimal one in which an MIDS is computed. By the safe convergence property, each configuration from a feasible one to an optimal one in the computation keeps the safety property.

Since we assume a synchronous scheduler, neighbor processes may update their local states simultaneously when a process updates its local state. Difficulty of designing such an algorithm comes from this concurrency. Each process can update its local state only when a safety property is not broken. On the other hand, when a configuration is feasible but not optimal, at least one process must make a move for convergence. Thus, designing an algorithm that satisfies these two requirements is not trivial under a synchronous scheduler. (If we assume a serial scheduler, the problem is trivial since there is no concurrency in process execution.)

Although we adopt process identifiers for priority scheme in this paper, we can adopt various priority (weighting) scheme, e.g., stability of a mobile node based on battery power and quality of wireless communication. Especially, a safely converging self-stabilizing system has a noble property in a situation such that process priorities dynamically changes. Consider a mobile ad hoc network in which stability of each node dynamically changes, and suppose that we adopt stability for priority scheme. Despite such a dynamic situation, configuration gracefully moves to optimal one with keeping a safety property.

1.2. Related works

Safety and stabilization

There are many works on extensions of self-stabilization for quick convergence and guarantee of safety property. In [3], concept of superstabilization is proposed for dynamic network. When a topology changes (e.g., a process joins or leaves a network) and the system is in a legitimate configuration, the system gracefully and quickly converges to a configuration. In [6], concept of fault-containment is proposed. Suppose that states of f processes are corrupted when a system is in a legitimate configuration. Then, a fault-containing system converges in $O(f)$ steps.

In [9, 8], the maximum flow routing on tree networks is proposed with the root process. The algorithm dynamically maintains the maximum flow route from each process to the route based on the current connection that dynamically change. In this sense, it maintains a safety after it stabilizes. The algorithm executes, in turn, a round for computing the correct flow and a round for updating a maximum flow tree. The root process controls execution of rounds so that each round is executed for enough long period of time to stabilize.

Unfortunately, above works consider only keeping safety in the event of faults or changes of system parameters in a legitimate configuration. That is, safety in converging configurations is not considered.

In [5], a concept safe stabilization is proposed. A safe self-stabilizing system guarantees that, for some given constant k, any k faults in a safe configuration does not lead to an unsafe configuration. Although a safe stabilizing system guarantees safety in convergence and thus very interesting, execution of such a system has a large overhead. It requires $\Omega(D)$ steps so that each process makes a single move, where D is the diameter of a network.

On the other hand, our framework does not guarantee safety when faults occur in legitimate configuration, i.e., our framework does not mask faults in legitimate configuration. By this property, our framework proposed in this paper does not require any overhead, and implementation is much easier. Thus we believe that our framework is worth investigating.

The most related works to our current paper are [1] and [12]. In [1], a stabilizing loop-free routing protocol is proposed. Starting from illegitimate configuration, eventually configuration reaches a safe configuration in which routes contains no loops. Then, configuration reaches, with maintaining loop-less property, a configuration in which routes with maximal metric. The work in [12] also proposes a stabilizing routing algorithm with route preserving property such that any message sent to the root node is guaranteed to be delivered within a finite time regardless continuous changes of link costs. Starting from illegitimate configuration, configuration eventually reaches one with route preserving property. Then, with keeping this as a safety property, the protocol obtains the shortest path. In this paper, we formulate a concept implicit in these papers.

The minimal dominating set (MDS) problem

Because the MDS problem is a fundamental prob-

lem on network topology, many self-stabilizing algorithms have been proposed so far. As a closely related problem, the maximal independent set (MIS) problem has been studied well. In [13], a self-stabilizing MIS algorithm under a serial scheduler (c-daemon)[1] is presented. In [11], a space-optimal self-stabilizing MIS algorithm under a distributed scheduler (d-daemon)[2] is presented. In [14], a self-stabilizing MDS algorithm under a synchronized scheduler is proposed. In [7], a self-stabilizing MIS algorithm under a synchronized scheduler is proposed. In [10] self-stabilizing MIS and MDS algorithms under a serial scheduler are proposed.

1.3. Organization of this paper

This paper is organized as follows. In section 2, we formally describe system model and safe self-stabilization. In section 3, we propose a safely self-stabilizing algorithm for the dominating set problem. In section 4, we show proof of correctness of the proposed algorithm, and show performance analysis. In section 5, we give conclusion of this paper and discuss future works.

2. Preliminary

2.1. System model

Let $V = (P_1, P_2, ..., P_n)$ be a set of processes and $E \subseteq V \times V$ be a set of bidirectional communication links in a distributed system. The number of processes is denoted by n. Then, the topology of the distributed system is represented as a undirected graph $G = (V, E)$. We assume that the graph is connected and simple. In this paper, we use "graphs" and "distributed systems" interchangeably.

A set of local variables defines local state of a process. By Q_i, we denote local state of process $P_i \in V$. A tuple of local state of each process $(Q_1, Q_2, ..., Q_n)$ forms a *configuration* of a distributed system. Let Γ be a set of all configurations.

As communication model, we assume that each process can read local states of neighbor processes without delay. This model is called the *state-reading model*. Each process can update its own local state only, but each process can read local state of neighbor processes.

An algorithm of each process P_i is given as a set of guarded commands (GCs):

$$*[\ Guard_1 \rightarrow Act_1 \ \square\ Guard_2 \rightarrow Act_2 \ \square\ Guard_3 \rightarrow Act_3 \ \square \cdots]$$

Each $Guard_j$ $(j = 1, 2, ...)$ is called a *guard*, and it is a predicate on P_i's local state and local states of its neighbors. Each Act_j is called a *action*, and it updates local state of P_i; next local state is computed from current local states of P_i and its neighbors. For each process P_i, process identifier and a set of neighbor processes N_i are given as a constant. We say that P_i is *privileged* in a configuration γ if and only if at least one guard of P_i is true in γ. An atomic step of each process P_i consists of the following three sub-steps (1) read local states of neighbor processes and evaluate guards, (2) execute a command that is associated to a true guard, and (3) update its local state. In this paper, we that assume the *synchronized model* for process execution such that every privileged process executes atomic steps in parallel.

2.2. Self-stabilization and safe convergence

For any configuration γ, let γ' be any configuration that follows γ. Then, we denote this transition relation by $\gamma \rightarrow \gamma'$. For any configuration γ_0, a *computation* E starting from γ_0 is a maximal (possibly infinite) sequence of configurations $E = \gamma_0, \gamma_1, \gamma_2, ...$ such that $\gamma_t \rightarrow \gamma_{t+1}$ for each $t \geq 0$.

Definition 1 (Self-stabilization) *Let Γ be the set of all configurations. A system S is* self-stabilizing *with respect to Λ such that $\Lambda \subseteq \Gamma$ if and only if it satisfies the following two conditions:*

- *Convergence: Starting from an arbitrary configuration, a configuration eventually becomes one in Λ, and*
- *Closure: For any configuration $\lambda \in \Lambda$, any configuration γ that follows λ is also in Λ.*

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

Definition 2 (Safely converging self-stabilization) *Let Γ be the set of all configurations, and let $\Lambda_O \subseteq \Lambda_F \subseteq \Gamma$. A self-stabilizing system S is* safely converging *with respect to (Λ_F, Λ_O) if and only if it satisfies the following three conditions:*

- *S is self-stabilizing with respect to Λ_F.*
- *Safe Convergence: For any execution starting from configuration in Λ_F, configuration eventually reaches one in Λ_O.*
- *S is self-stabilizing with respect to Λ_O.*

[1] At each step, a serial scheduler selects a process arbitrarily from a set of eligible processes.

[2] At each step, a distributed scheduler selects a non-empty set of processes arbitrarily from a set of eligible processes.

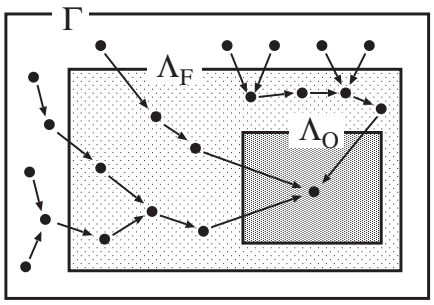

Figure 1. Safely converging self-stabilization: Γ, Λ_F and Λ_O

Each $\gamma \in \Lambda_F$ is called a feasibly legitimate *configuration, and each $\gamma \in \Lambda_O$ is called a* optimally legitimate *configuration.* □

Concept of safe converging self-stabilization is depicted in Figure 1. Time complexity of a safe self-stabilizing algorithm is measured by the time to reach Λ_F and the time to reach Λ_O. Formal definition is as follows.

Definition 3 *Let S be a safely converging self-stabilizing system with respect to (Λ_F, Λ_O). The* first convergence time *is the number of steps to reach a configuration in Λ_F for any starting configuration in Γ. The* second convergence time *is the number of steps to reach a configuration in Λ_O for any starting configuration in Λ_F.* □

2.3. Minimal independent dominating set

Minimal dominating set (MDS) is formally defined as follows.

Definition 4 *A* dominating set *of a graph $G = (V, E)$ is a subset $V' \subseteq V$ such that, for any $u \in V \backslash V'$, there exists $v \in V'$ such that $(u, v) \in E$. A dominating set V' of G is* minimal *if no proper subset of V' is a dominating set of G.* □

Maximal independent set (MIS) is defined as follows.

Definition 5 *An* independent set *of a graph $G = (V, E)$ is a subset $V' \subseteq V$ such that, for any $u, v \in V'$, $(u, v) \notin E$. An independent set V' of G is* maximal *if no proper superset of V' is an independent set of G.* □

It is clear that any MIS is also a MDS. Minimal independent dominating set (MIDS) is defined as follows.

Definition 6 *An* minimal independent dominating set *of a graph $G = (V, E)$ is a subset $V' \subseteq V$ such that V' is a minimal dominating set, and V' is an independent set.* □

3. Proposed Algorithm

Figure 2 shows a minimal independent dominating set algorithm we propose. In the proposed algorithm, each process P_i uses two variables d_i and m_i as described as follows.

- $d_i \in \{0, 1\}$: — This value is 1 (resp. 0) if P_i is a *dominator* (resp. *dominatee*).
- $m_i \in N_i \cup \{P_i\}$: — When P_i is a dominatee, it uses this variable to designate a neighbor process to dominate P_i by assignment $m_i = P_j (\in N_i)$.

  Assignment $m_i := P_j (\in N_i)$ does not imply that P_i is a dominatee. We have $m_i = P_j (\in N_i)$ and $d_i = 1$ when P_i wants to turn to be a dominatee from a dominator. This is for safety, and details will be explained later.

Let us explain idea of the proposed algorithm which is shown in Figure 2. As shown in Figure 2, we use a macro $IndDom_i(P_j)$ for each process P_i and its neighbor $P_j \in N_i$. It is defined to be true if and only if $(d_j = 1) \wedge (m_j = P_j)$ is true. That is, P_j is a dominator and it does not designate its neighbor to dominate P_j. We say that such a process P_j is an *independent dominator*, and $IndDom_i(P_j)$ is true.

Each process P_i changes its local variables according to the following rules.

- **Rule 1.** When identifier of P_i is the largest among those of its neighbors, P_i becomes an independent dominator.
- **Rule 2.** When P_i has no independent dominators in its neighbors, P_i becomes an independent dominator.
- **Rule 3.** When identifier of P_i is the largest among those of independent dominators in its neighbors, P_i becomes an independent dominator.
- **Rule 4.** When identifier of P_i is not the largest among those of independent dominators in its neighbors, the value of m_i is maintained to be the largest among them. For safety, P_i becomes a dominator.

external variable
process set N_i; — a dynamic set of neighbor processes (read only and automatically updated).
variables
int $d_i \in \{0, 1\}$; — P_i is a dominator (resp. dominatee) if $d_i = 1$ (resp. 0).
process name $m_i \in N_i \cup \{P_i\}$; — a dominator process that P_i depends on. The value can be P_i itself.
macro
$MaxLocally_i \equiv \forall P_j \in N_i : P_i > P_j$ — True iff the process identifier is the maxmimum among neighbors.
$IndDom_i(P_j) \equiv (d_j = 1) \wedge (m_j = P_j)$ — True iff a neighbor P_j is an independent dominator.
$ExNeighIndDom_i \equiv \exists P_j \in N_i : IndDom_i(P_j)$ — True iff there exists an independent dominator.
$MaxNeighIndDom_i \equiv \max\{P_j \in N_i : IndDom_i(P_j)\}$ — The maximum independent dominator in neighbors.
$NoneDepends_i \equiv \forall P_j \in N_i : m_j \neq P_i$ — True iff no neighbor depends on P_i.
actions
* [
Rule 1: If identifier of P_i is the locally largest, P_i becomes an independent dominator.
$MaxLocally_i$
$\wedge((d_i \neq 1) \vee (m_i \neq P_i))$
$\rightarrow \quad d_i := 1;\ m_i := P_i;$
Rule 2: If there is no independent dominator in neighbors, P_i becomes an independent dominator
□ $\neg MaxLocally_i \wedge \neg ExNeighIndDom_i$
$\wedge((d_i \neq 1) \vee (m_i \neq P_i))$
$\rightarrow \quad d_i := 1;\ m_i := P_i;$
Rule 3: If identifier of P_i is the largest among independent dominators in neighbor, P_i becomes an independent dominator.
□ $\neg MaxLocally_i \wedge ExNeighIndDom_i \wedge (P_i > MaxNeighIndDom_i)$
$\wedge((d_i \neq 1) \vee (m_i \neq P_i))$
$\rightarrow \quad d_i := 1;\ m_i := P_i;$
Rule 4: If identifier of P_i is not the largest among independent dominators in neighbor, P_i depends on the process with the largest identifier among them, and (for safety) P_i becomes a dominator.
□ $\neg MaxLocally_i \wedge ExNeighIndDom_i \wedge (P_i < MaxNeighIndDom_i) \wedge (m_i \neq MaxNeighIndDom_i)$
$\wedge((d_i \neq 1) \vee (m_i \neq MaxNeighIndDom_i))$
$\rightarrow \quad d_i := 1;\ m_i := MaxNeighIndDom_i;$
Rule 5: P_i turns to be a dominatee only when (1) identifier is not the largest among independent dominators in neighbors, (2) P_i depends on the process with the largest identifier among them, and (3) (for safety) no neighbor depends on P_i.
□ $\neg MaxLocally_i \wedge ExNeighIndDom_i \wedge (P_i < MaxNeighIndDom_i) \wedge (m_i = MaxNeighIndDom_i) \wedge NoneDepends_i$
$\wedge(d_i \neq 0)$
$\rightarrow \quad d_i := 0;$
]

Figure 2. SC-MIDS**: A safely converging self-stabilizing algorithm for MIDS**

- **Rule 5.** P_i turns to be a dominatee if there exists an independent dominator in its neighbor, provided that the value of m_i is property maintained by Rule 4 and, for safety, no neighbor $P_j \in N_i$ designates P_i to dominate P_j.

Let Γ be a set of all configurations. First, we define *Doms* as follows.

Definition 7 *For each configuration $\gamma \in \Gamma$, we define $Doms(\gamma) \equiv \{P_i \in V : d_i = 1\}$, which is called a* set of dominator processes *in γ.* □

Sets of legitimate configurations Λ_F and Λ_O of the proposed algorithm are defined as follows.

Definition 8 *A set of* feasibly legitimate *configurations $\Lambda_F \subseteq \Gamma$ is defined as follows.*

$$\Lambda_F = \{\gamma \in \Gamma : \forall P_i \in V : (d_i = 0) \Rightarrow (m_i \in N_i \wedge d_j = 1, \text{ where } P_j = m_i \text{ in } \gamma)\}$$

A set of optimally legitimate *configurations $\Lambda_O \subseteq \Lambda_F$ is defined as follows.*

$$\Lambda_O = \{\gamma \in \Lambda_F : Doms(\gamma) \text{ is the lexicographically first minimal independent dominating set of } G\} \quad \square$$

Note: Although Λ_O is called *optimal*, a configuration in the set is *not* the *minimum* independent dominating set, but it is *minimal*.

By definition of Λ_F, it is clear that $Doms(\gamma)$ is a dominating set of G for any $\gamma \in \Lambda_F$. Further, Λ_F is a set of configuration in which a dominating set is computed, and for each dominatee P_i, its dominator P_j designated by m_i is a dominator. Because Λ_O is the lexicographically first minimal dominating set of G, it contains only one configuration. For simplicity of notation in the proof, we say that "a configuration γ is a dominating set" if $Doms(\gamma) = \{P_i \in V : d_i = 1\}$ is a dominating set of G.

4. Proof of Correctness

In this section, we show proof of correctness of the proposed algorithm. In the proof we use I_i for each $P_i \in V$ which is defined as follows.

$$I_i \equiv (d_i = 0) \Rightarrow (m_i \in N_i \wedge d_j = 1, \text{ where } P_j = m_i)$$

The relation of I_i and Λ_F is as follows.

$$\gamma \in \Lambda_F \Leftrightarrow \forall P_i \in V : I_i \text{ in } \gamma$$

Lemma 1 ***(One step convergence to*** Λ_F***)*** *Let γ be any configuration in Γ, and γ' be a configuration such that $\gamma \to \gamma'$. Then, we have $\gamma' \in \Lambda_F$.*

Proof: It is enough to show that I_i holds at each process $P_i \in V$ in γ'. As a contrary, we assume that there exists a process P_i in γ' such that $\neg I_i$.

- In case P_i is not executed in γ:

  Since P_i is not executed and I_i can be false only when $d_i = 0$, guards of Rules 1–4 must be false and $d_i = 0$ in γ. (Note that a process that has a true guard is executed by assumption of the execution model.) Since $d_i = 0$ in γ', the following conditions hold in γ:

  - $\neg MaxLocally_i$ (by Rule 1)
  - $ExNeighIndDom_i$ (by Rule 2)
  - $P_i < MaxNeighIndDom_i$ (by Rule 3)
  - $m_i = MaxNeighIndDom_i$ (by Rule 4)

  Then, $m_i \in N_i$ holds in γ, and thus we have $d_j = 1$, where $P_j = m_i$. This implies I_i in γ' and a contradiction.

- In case P_i is executed in γ:

  Because $d_i = 0$ in γ', P_i is executed Rule 5 in γ. (Otherwise, any other rule results in $d_i = 1$.) Then $m_i = MaxNeighIndDom_i$ is true in γ because it is a part of the condition of Rule 5. This implies that $m_i \in N_i$ in γ. Since Rule 5 does not change the value of m_i, we have $m_i \in N_i$ in γ'.

  Let P_j be a process given by the value of m_i. By definition of $MaxNeighIndDom_i$, $d_j = 1 \wedge m_j = P_j$ holds in γ. Because $NoneDepends_j$ is a part of the condition of Rule 5, P_j never execute Rule 5 in γ, and thus $d_j = 1$ in γ'. This implies I_i holds in γ' and a contradiction.

Since there exists no P_i such that $\neg I_i$ in γ', we have $\gamma' \in \Lambda_F$. □

Lemma 2 ***(Closure of*** Λ_F***)*** *Let γ be any configuration in Λ_F, and γ' be any configuration such that $\gamma \to \gamma'$. Then, we have $\gamma' \in \Lambda_F$.*

Proof: By Lemma 1, this lemma trivially holds. □

Lemma 3 ***(Convergence to*** Λ_O***)*** *Let γ be any configuration in Λ_F. For any execution starting from γ, configuration γ' reaches one such that $\gamma' \in \Lambda_O$, and configuration never changes thereafter.*

Proof: Let $\gamma_0 = \gamma$, and for each $t \geq 1$, let γ_t be the t-th configuration in a computation starting from γ. Note that γ_t be the configuration by execution of the t-th step. For simplicity of explanation, for each integer $x \geq 1$, we call four steps from the $4x-3$-th step to the $4x$-th step as the *x-th phase*.

1. The first phase (i.e., from the 1st to the 4th steps).

   - The 1st step: We focus on the process, say P_M, with the maximum process identifier in the network G. We have $d_M = 1 \wedge m_M = P_M$ by Rule 1 in γ_1. Note that P_M may or may not execute Rule 1 in γ_0. If $d_M = 1 \wedge m_M = P_M$ is false, P_M executes Rule 1, and otherwise, it updates none of these variables. Since Rule 1 is the only rule that P_M may execute, P_M never execute thereafter, and the variable values of P_M are fixed in this configuration.
   - The 2nd step: We focus on neighbor processes of P_M. Since $MaxNeighIndDom_j = P_M$ and $P_j < P_M$ for each $P_j \in N_M$, by Rule 4, we have $P_j = 1$ and $m_j = P_M$ in γ_2. Since P_M has the maximum process identifier, each $P_j \in N_M$ never change the value of m_j thereafter.
   - The 3rd step: We focus on neighbor P_k of each neighbor of P_M. Since $m_j \neq P_j$ for each $P_j \in N_M$, each $P_k \in (N_j \backslash N_M \backslash \{P_M\})$ selects some P_ℓ not in N_M by Rule 3 and 4.

- The 4th step: Again we focus on neighbors of P_M. Let P_j be any neighbor of P_M. Then, by executions of processes in the 3rd step, we have $m_k \neq P_j$ for each process $P_k \in P_j$. Thus P_j executes Rule 5. Since P_M has the maximum process identifier, each P_j never change the values of d_j and m_j thereafter. Note that the value of d_j (resp. m_j) is 0 (resp. P_M).

In the first phase, the process with the maximum process identifier and its neighbors decide their variable values, and they never change their local variables thereafter.

When the first phase is finished, each $P_j \in N_M$ is not an independent dominator because $d_j = 0 \wedge m_j \neq P_j$ holds.

Let us observe behavior of processes after the first phase. By Rule 4, when P_i selects the value of m_i from neighbors, it selects an independent dominator with the largest process identifier among independent dominators in neighbors. By Rule 5, each process P_i becomes a dominatee only when it is a dominator, and the value of m_i has the largest process identifier among independent dominators in neighbors. Because of these rules, each $P_k \in (V \backslash N_M \backslash \{P_M\})$ never select $P_j \in N_M$ and P_M for the value of m_k. Therefore, we can ignore P_M and processes in N_M in the following execution.

Let G_1 be a graph by deleting processes $\{P_M\} \cup N_M$ and corresponding edges from G. Next, we consider graph G_1.

2. The second phase (i.e., from the 5th to the 8th steps).

   - The 5th step: We focus on the process, say $P_{M'}$ with the maximum process identifier in G_1. By Rule 3, we have $d_{M'} = 1 \wedge m_{M'} = P_{M'}$ in γ_5, because process identifier of $P_{M'}$ is larger than any independent dominators in neighbors in G_1. Note that P_M (discussed in the first phase) is not a neighbor of $P_{M'}$ in G. Then, the values of $d_{M'}$ and $m_{M'}$ never change thereafter.
   - The 6th step: The same as the 2nd step.
   - The 7th step: The same as the 3rd step.
   - The 8th step: The same as the 4th step.

   When the second phase is finished, as in the case of the first phase, the process with the maximum process identifier in G_1 becomes a dominator, its neighbors become dominatees, and variables values of them never change in the following execution. Let G_2 be the graph by deleting these processes and corresponding edges.

3. The third and following phases.

   We repeatedly apply the same discussion for the second phase. In each phase $t > 2$, the process with the maximum process identifier in G_{t-1} becomes a dominator and its neighbors become dominatees, and graph G_t is defined. (This discussion is repeated as long as graph is not empty.)

It is clear that at least one process is eliminated at each phase, graph becomes empty within n phases. Let γ' be the corresponding configuration when a graph becomes empty. Then, each condition of the rules becomes false at each process, and no process makes a move thereafter.

Since, in γ', no two dominators are neighbors each other and each dominatee has a dominator in neighbors, $Doms(\gamma')$ is an MIDS of $G(V, E)$. It is clear that $Doms(\gamma')$ is the lexicographically first one by the discussion above. □

Theorem 1 *SC-MIDS is safely converging self-stabilizing with respect to (Λ_F, Λ_O), and the first (resp. second) convergence time is at most 1 (resp. $O(D)$), where D is the diameter of network.*

Proof: Convergence to Λ_F is shown by Lemma 1, and convergence time to Λ_F is one. Closure of to Λ_F is shown by Lemma 2. Hence, SC-MIDS is self-stabilizing with respect to Λ_F. Safe convergence property is shown by Lemma 3. Convergence to Λ_O and closure of to Λ_O are shown by Lemma 3. This fact and Lemma 1 prove that SC-MIDS is self-stabilizing with respect to $\Lambda_F O$.

Let us derive convergence time. By Lemma 1, the first convergence time is one. From the proof of Lemma 3, we observe that a dominator and at least one dominatee are eliminated at each phase except the last one. Thus the second convergence time is trivially $O(n)$. We show below that the second convergence time is in fact $O(D)$.

Observe the proof for convergence (Lemma 3). In each phase, there is at least one process that is elected as an independent dominator and the values of local variables are surely fixed forever. For each $t = 1, 2, 3, ...$, let $X_t \subseteq V$ be a set of such processes at phase t. Note that $X_t \cap X_{t'} = \emptyset$ for each t and t' such that $t \neq t'$, and a set $\bigcup_t X_t$ is the lexicographically first independent dominating set of G.

Let t be any phase such that $t > 1$. For each process P_i in X_t to be elected as a dominator, the following conditions hold.

- P_i is not a process with the locally largest process identifier. (Otherwise, P_i would be elected in phase 1 by Rule 1.)

- At least one of $P_j \in N_i$ such that $P_j > P_i$ is a dominator at the beginning of phase $t-1$, every such P_j turns to be dominated by some $P_k \in X_{t-1}$ at the end of phase $t-1$, and each neighbor $P_j \in N_i$ such that $P_j > P_i$ is a dominatee at the beginning of phase t. (Otherwise, P_i would not be in X_t.)

- P_i is a process with the locally largest process identifier among independent dominators in neighbors in phase t. (Otherwise, P_i would not be in X_t.)

Thus, the decision of a process in X_t at phase t depends on the decisions of processes in X_{t-1} and its neighbors at phase $t-1$. It is clear that the number of phases to converge is determined by the length of such a dependency which is at most D, the diameter of the network. Therefore, D phases are enough to converge, and the second convergence time is $O(D)$. □

5. Conclusion

In this paper, we proposed a concept of safe convergence in self-stabilization and an algorithm for the minimal independent dominating set (MIDS) problem that computes the lexically first one. By our algorithm, a system converges to a configuration of a dominating set in one step, and it converges to a configuration of a minimal dominating set in $O(D)$ steps, where D is the diameter of a network.

We adopted process identifiers as a priority for selecting a dominator for simplicity. In fact, we can adopt other metrics as a priority, such as the number of neighbors (i.e., $|N_i|$), battery power, and computational power for example. Process identifiers are used for breaking a tie.

We assumed the synchronized model for execution and state-reading model for communication in this paper. We believe that our algorithm is correct under the distributed scheduler in which arbitrary set of processes is selected to execute. Extension of the algorithm for the message passing model is left for future task.

References

[1] J. Cobb and M. Gouda. Stabilization of general loop-free routing. 62:922–944, 2002.

[2] E. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.

[3] S. Dolev and T. Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago J. of Theoretical Comput. Sci.*, 3(4), 1997.

[4] F. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31:1–26, 1999.

[5] S. Ghosh and A. Bejan. A framework of safe stabilization. In *Proceedings of SSS, LNCS 2704*, pages 129 – 140, 2003.

[6] S. Ghosh, A. Gupta, T. Herman, and S. Pemmaraju. Fault-containing self-stabilizing algorithms. In *Proceedings of PODC*, pages 45–54, 1996.

[7] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Self-stabilizing protocols for maximal matching and maximal independent sets for ad hoc networks. In *Proceedings of IPDPS APDCM*, page 162b, 2003.

[8] M. Gouda and M. Schneider. Stabilization of maximum flow trees. In *Proceedings of the Annual Joint Conf. on Inform. Sci.*, pages 178–181, 1994.

[9] M. Gouda and M. Schneider. Maximum flow routing. In *Proceedings of WSS*, pages 2.1–2.13, 1995.

[10] S. M. Hedetniemi, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Self-stabilizing algorithms for minimal dominating sets and maximal independent set. *Computer Mathematics and Applications*, 2003.

[11] M. Ikeda, S. Kamei, and H. Kakugawa. A space-optimal self-stabilizing algorithm for the maximal independent set problem. In *Proceedings of PDCAT*, pages 70 – 74, 2002.

[12] C. Johnen and S. Tixeuil. Route preserving stabilization. In *Proceedings of SSS, LNCS 2704*, pages 184–198, 2003.

[13] S. Shukla, D. Rosenkrantz, and S. Ravi. Observation on self-stabilizing graph algorithms for anonymous networks. *Proceedings of WSS*, 1995.

[14] Z. Xu, S. T. Hedetniemi, W. Goddard, and P. K. Srimani. A synchronous self-stabilizing minimal domination protocol in an arbitrary network graph. In *Proceedings of IWDC, LNCS 2918*, pages 26 – 32, 2003.