PAPER Special Section on Foundations of Computer Science

A Self-Stabilizing Distributed Algorithm for the Steiner Tree Problem*

Sayaka KAMEI†, ***Student Member and*** **Hirotsugu KAKUGAWA**†, ***Nonmember***

SUMMARY Self-stabilization is a theoretical framework of non-masking fault-tolerant distributed algorithms. In this paper, we investigate the Steiner tree problem in distributed systems, and propose a self-stabilizing heuristic solution to the problem. Our algorithm is constructed by four layered modules (sub-algorithms): construction of a shortest path forest, transformation of the network, construction of a minimum spanning tree, and pruning unnecessary links and processes. Competitiveness is $2(1 - 1/l)$, where l is the number of leaves of optimal solution.

key words: *distributed algorithm, fault-tolerant, self-stabilizing algorithm, fair composition, Steiner tree problem*

1. Introduction

1.1 Self-Stabilization

A distributed system is a set of processes and a set of communication links between the processes. Because distributed systems are subject to fail by their very nature, *fault-tolerance* is a major concern in the study of distributed computing.

Fault-tolerant systems are classified into two categories: masking and non-masking [6]. If liveness property is guaranteed, but safety property is not guaranteed in the presence of faults, it is called non-masking. Self-stabilization is a theoretical framework of non-masking fault-tolerant distributed algorithms proposed by E.W. Dijkstra [4]. Self-stabilizing algorithms can start execution from arbitrary (illegitimate) configuration and eventually reach a legitimate configuration. By this property, self-stabilizing algorithms tolerate any kind and any finite number of transient faults [5].

1.2 The Steiner Tree Problem

Let $G = (V, E, w)$ be a weighted graph, where V is a set of nodes, E is a set of edges, and $w : E \to \mathbb{R}$ is a cost (weight) function. We assume that G is undirected and connected. It is well known that a minimum cost spanning tree of G has many practical applications. In a spanning tree G_T of G, every node of G is involved in G_T. In some application, not all nodes need to be involved. Let $Z \subseteq V$ be any set of nodes. A minimum cost subgraph G_z of G that spans Z is called a *Steiner* tree for Z. When $|Z| = 2$, G_z is the shortest path between two nodes in Z, and when $|Z| = n$, where n is the number of nodes on G, G_z is a minimum spanning tree.

Because it is known that the problem of computing a minimum cost Steiner tree for nontrivial Z is NP-complete [8], many heuristic algorithms have been proposed as a practical alternative. In many heuristic algorithms, a minimum cost spanning tree is computed first and then unnecessary nodes and edges in the tree are pruned. Such a heuristic is called Pruned-MST [15], and its competitiveness** is $n-|Z|+1$. Because competitiveness of Pruned-MST is not good, it is combined with other heuristics as a building block.

Kou et al. proposed a heuristic algorithm in [9]. In [16], Wu et al. proposed an improved version of this algorithm whose competitiveness is $2(1 - 1/l) < 2$, where l is the number of leaves in the optimal Steiner tree for G and Z. Many other sequential heuristic algorithms have been proposed whose competitiveness tends to 2, notably by Takahashi et al. [13], Rayward-Smith [11] and Mehlhorn [10]. Also Robins et al. improved competitiveness from 2 to 1.55, e.g. [12], after a long period without any progress. The literatures [15] and [7] are good surveys for this problem.

In distributed systems, the problem of optimal routing of multicast, telecommunications and so on, can be modeled by the Steiner tree problem. Let $G = (V, E, w)$ be a network topology, where V is a set of processes, E is a set of communication links, and w is a cost function. A Steiner tree for $Z \subseteq V$ is a minimum cost multicast routing on G when Z is a set of a source and destination nodes. Since no efficient algorithm is known for the Steiner tree problem, a minimum spanning tree is used often as an approximation of a Steiner tree in practice, e.g. DVMRP [14].

In [2], Chen et al. proposed a distributed version of the sequential algorithm by Wu et al. [16]. They assume asynchronous message passing model with no failures. Because the algorithms in [16] and [2] are essentially the same, their competitiveness are the same. Because it is not easy to convert sequential algorithms with good competitiveness such as [12] into distributed algorithms, there is no other dis-

Manuscript received February 24, 2003.
Manuscript revised July 8, 2003.

†The authors are with the Department of Information Engineering, Hiroshima University, Higashi-Hiroshima-shi, 739–8527 Japan.

*This work is partially supported by the Ministry of Education, Culture, Sports, Science and Technology under grant No.15700017. Preliminary version of a part of this paper was presented at International Workshop on Self-Repairing and Self-Configurable Distributed Systems, 2002.

**Competitiveness is a ratio C_{alg}/C_{opt}, where C_{alg} is the cost of the solution of the approximation algorithm in the worst case and C_{opt} is the cost of the optimal solution.

tributed algorithms for the Steiner tree problem to the best of our knowledge.

1.3 Contribution of This Paper

In distributed systems, fault-tolerance is one of the most important issues, and dynamic changes of a system (such as the members of Z) should be taken into account.

In this paper, we consider fault-tolerance in computing a Steiner tree, and we propose a self-stabilizing heuristic distributed algorithm based on the idea shown in [2]. Our algorithm is truly distributed in a sense that there is no centralized process to compute a Steiner tree. By the properties of self-stabilization, the algorithm is also adaptive to dynamic Steiner tree problem in which the members of Z changes dynamically. To the best of our knowledge, our algorithm is the first self-stabilizing distributed algorithm for the Steiner tree. Competitiveness of our algorithm is $2(1 - 1/l)$, where l is the number of leaves in the optimal Steiner tree, which is the same as ones in [2], [9], [16].

This paper is organized as follows. In Sect. 2, we give formal definitions of the self-stabilizing distributed Steiner tree problem and a computational model assumed in this paper. In Sect. 3, we present an outline of a non-self-stabilizing distributed algorithm of Chen et al.,[2] on which our algorithm is based. In Sect. 4, we propose a self-stabilizing algorithm for the Steiner tree problem, and we give proofs of correctness for the proposed algorithm. In Sect. 5, we give concluding remarks.

2. Preliminaries

2.1 The System Model

Let $V = (P_1, P_2, \ldots, P_n)$ be a set of processes, where n is the number of processes, and $E \subseteq V \times V$ be a set of bidirectional communication links in a distributed system. We define cost for each link. It is given by a cost function $w : E \to \mathbb{R}$ which maps E into the set of non-negative numbers. Then, the topology of the distributed system is represented as an undirected weighted graph $G = (V, E, w)$. We assume that the graph is connected and simple. In this paper, we use "graphs" and "distributed systems" interchangeably. By N_i, we denote a set of neighbor processes of P_i, and by Δ_i, we denote the degree of P_i, i.e., $\Delta_i = |N_i|$.

Let the *distance* of the path be the sum of the costs of the edges on the path, and $\mathrm{dist}(P_i, P_j)$ be the distance of the shortest path between P_i and P_j.

We assume that each process P_i has unique process identifier. Let *id* be a naming function of processes. By $id(P_i)$, we denote the process identifier of P_i for each process P_i. In discussing process identifier, with abuse of notation, we use P_i to denote $id(P_i)$ when it is clear from context.

As a communication model, we assume that each process can read local states of neighbor processes without any delay. This model is called the *state reading model*. It is also known as the *shared memory model*. Although a process can read local states of neighbor processes, it cannot update them; it can update only local state of itself.

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 $(q_1, q_2, \ldots, q_n)$ forms a *configuration* of a distributed system. Let Γ be a set of all configurations.

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

$$*[\ g_1 \to c_1;\ g_2 \to c_2;\ g_3 \to c_3;\ \cdots\ ]$$

Each g_j $(j = 1, 2, \ldots)$ is called a *guard*, and it is a predicate on P_i's local state and local states of its neighbors. Each c_j is called a *command*, 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 parameters, and these values are available in guarded commands. We say that P_i is *privileged* in 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.

Execution of processes are scheduled by an external (virtual) scheduler. A scheduler decides which process to execute in the next step. At each step, a scheduler selects only one privileged process arbitrarily, and a selected process executes an atomic step. This type of scheduler is known as the *central daemon*. A scheduler is *fair* if privileged process is eventually selected to execute by a scheduler, otherwise, it is *unfair*. We assume the centralized scheduler is unfair in this paper. Because a privileged process may not be executed forever under an unfair scheduler, an algorithm must be correct with respect to every execution scheduling. Thus, a scheduler is an adversary against an algorithm.

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

Definition 1: Let Γ be a set of all configurations, and Λ be a set of legitimate configurations. An algorithm A is *self-stabilizing* with respect to $\Lambda \subseteq \Gamma$ if and only if the following two conditions hold:

- Convergence: Starting from an arbitrary configuration, 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, and Λ is called a set of legitimate configurations. □

2.2 The Steiner Tree Problem

Generally, the Steiner tree problem is defined as follows.

Definition 2: Let $G = (V, E, w)$ be a weighted graph, where V is a set of nodes, E is a set of edges, and $w : E \to \mathbb{R}$ is an edge cost function. For any non-empty subset $Z \subseteq V$, $G_z = (V_z, E_z, w)$ is a *Steiner tree* for Z and G if and only if G_z is the minimum cost subgraph of G such that G_z is connected and $Z \subseteq V_z \subseteq V$ holds. □

We call a member of Z as a *Z-member*.

We consider solving the Steiner problem in distributed system in this paper. We assume that each process does not know global information of the network, and they know local information only. Under such assumption, we defined the distributed Steiner tree problem as follows.

Definition 3: The *distributed Steiner tree problem* is defined as follows.

- Given: Each process is given (1) a set of costs of the incident links and (2) if it is a Z-member or not.
- Find: Each process decides (1) if it is a member of a Steiner tree or not, and (2) for each incident edge e, if e is an edge of a Steiner tree or not. □

Note that G is a distributed system and each process in G must compute a Steiner tree of G itself.

3. Chen et al.'s Distributed Algorithm

Wu et al. proposed a sequential heuristic algorithm in [16], which is an improved version of the algorithm by Kou et al. [9]. In [2], Chen et al. proposed the distributed version of the algorithm by Wu et al., and basic idea of these algorithms ([16] and [2]) are essentially the same. Our self-stabilizing algorithm is based on these algorithms, and we present outline of their algorithms.

The essence of these algorithms is to find a *generalized minimum spanning tree* G_z for Z in G, which is an approximation of the minimum cost Steiner tree.

Definition 4: For $G = (V, E, w)$ and $Z \subseteq V$, let $G_c = (Z, E_c, w_c)$ be the complete weighted graph where $E_c = \{(P_i, P_j) \mid P_i, P_j \in Z, P_i \neq P_j\}$, and $w_c(P_i, P_j)$ is equal to $\mathrm{dist}(P_i, P_j)$ in G for each $P_i, P_j \in Z$ $(P_i \neq P_j)$. Let $G_s = (Z, E_s, w_c)$ be a minimum spanning tree of G_c. A *generalized minimum spanning tree* $G_z = (V_z, E_z, w)$ is a subgraph of G satisfying the following conditions.

- A node p is in V_z iff there exists an edge $(u, v) \in E_s$ such that p is a node on the shortest path between u and v (including u and v) in G.
- An edge e is in E_z iff there exists an edge $(u, v) \in E_s$ such that e is an edge on the shortest path between u and v in G.
- Each leaf of G_z is in Z. □

To compute a generalized minimum spanning tree, a shortest path forest is computed, which is defined as follows.

Definition 5: For $G = (V, E, w)$ and $Z \subseteq V$, a *shortest path forest* of G is a subgraph $G_F = (V, E_F, w)$ of G consisting of a set of disjoint trees $T_i = (V_i, E_i, w)$, $i = 1, 2, \ldots, |Z|$ that satisfies the following conditions.

- $\bigcup_{i=1}^{|Z|} V_i = V$ and $V_i \cap V_j = \emptyset$ for all $i \neq j$.
- $\bigcup_{i=1}^{|Z|} E_i = E_F \subseteq E$.
- For each $1 \le i \le |Z|$, V_i contains exactly one node $P_z^i \in Z$, and each $P_z^i \in Z$ appears in T_i.
- For each V_i, let P_z^i be a Z-member in V_i. Then, $P_j \in V_i$ iff $\mathrm{dist}(P_j, P_z^i) \le \mathrm{dist}(P_j, P_z)$ for any $P_z \in Z$,
- P_z^i is the root of T_i for each $1 \le i \le |Z|$. □

In a shortest path forest, edges of G are classified into the following three types:

- An edge e of G is a *tree edge* if it is also an edge of G_F.
- An edge e of G is an *intra-tree edge* if it is not in G_F and its endpoints are in the same tree of G_F.
- An edge e of G is an *inter-tree edge* if it is not in G_F and its endpoints are in different trees of G_F each other.

Figure 1 (a)–(e) illustrate an execution example. Black vertices are Z nodes, and edge labels are edge costs. Figure 1 (a) shows an underlaying network $G = (V, E, w)$. Intuitive outline of the algorithm is as follows.

First, in Step 1, we construct a shortest path forest G_F. Then each tree in a forest is a group of nodes that consists of a Z-member and some non-Z-members. Each non-Z-member belongs to a group such that the distance from the Z-member of a group is not larger than any other Z-members. Figure 1 (b) shows a shortest path forest G_F.

Next, in Step 2, we (virtually) contract nodes in a group (a tree) for each group, i.e., we regard a group as a virtual node. To preserve the distance between Z-members, we reassign the cost of edges that bridges two virtual nodes, i.e., inter-tree edges. Figure 1 (c) shows the corresponding graph G_P, where thick (resp. thin, dashed) edges are tree (resp. inter-tree, intra-tree) edges and edge labels are new costs. Then, in Step 3, a minimum spanning tree is computed for the contracted graph. Figure 1 (d) shows the minimum spanning tree where each tree on the shortest path forest is regarded as a node. Trees are connected by some selected inter-tree edges.

Finally, in Step 4, we prune every non-Z-member node which is a leaf in the graph obtained in the last step. Figure 1 (e) shows the Steiner tree obtained.

The outline is described more formally as follows.

Outline of the algorithm [2]

Step 1. Construct G_F which is a shortest path forest for Z in G. We assume that each node P_i of G knows its root $R(P_i)$ the distance of the shortest path from $R(P_i)$ to P_i, i.e. $\mathrm{dist}(P_i, R(P_i))$, and the father of P_i on this path.

Step 2. Construct $G_P = (V, E, w_P)$ from $G = (V, E, w)$

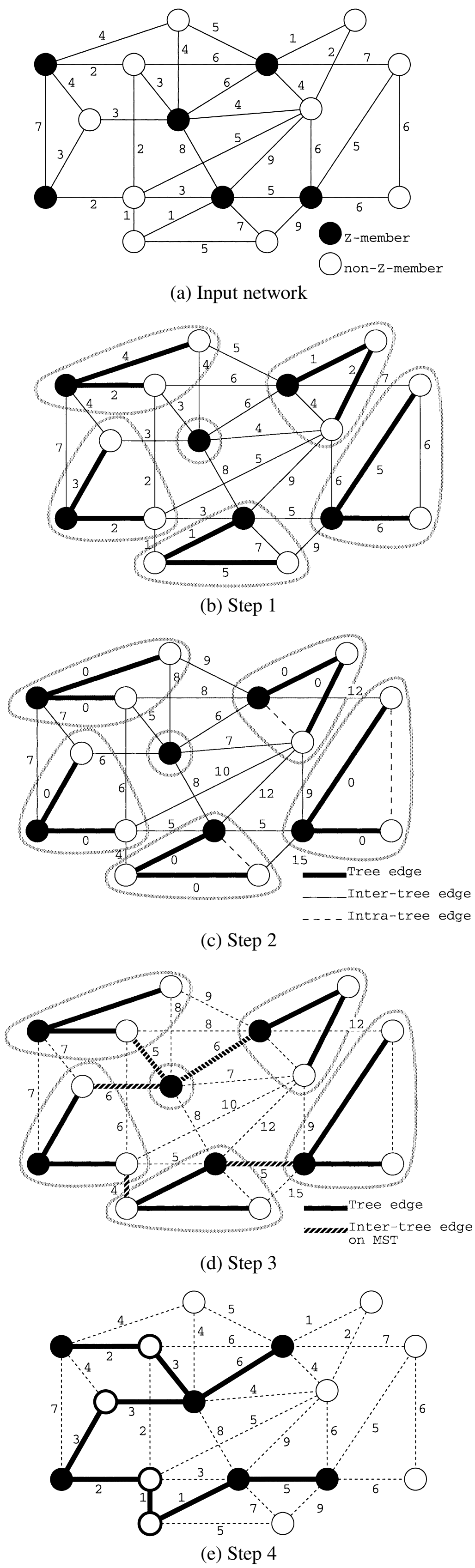

Fig. 1 An execution example.

and G_P which is the result in Step 1. The cost $w_P(P_i, P_j)$ is defined as follows:

Case 1: If (P_i, P_j) is a tree edge, $w_P(P_i, P_j) = 0$.
Case 2: If (P_i, P_j) is an intra-tree edge, $w_P(P_i, P_j) = \infty$.
Case 3: If (P_i, P_j) is an inter-tree edge, $w_P(P_i, P_j) = \mathrm{dist}(P_i, R(P_i)) + w(P_i, P_j) + \mathrm{dist}(P_j, R(P_j))$. This value is the distance of the shortest path via edge (P_i, P_j) between the roots of P_i and P_j in G.

Step 3. Construct G_M which is a minimum spanning tree of G_P.
Step 4. Construct G_z by pruning G_M. G_z is a generalized minimum spanning tree of G and Z.

4. The Proposed Self-Stabilizing Algorithm

In this section, we propose a layered self-stabilizing algorithm for the Steiner tree problem LSS-ST. The input is underlaying network G and a subset Z of V, and Z includes a source and a set of destinations. We assume that each process has a constant src_i which is true iff P_i is the source. The output is an approximation of the minimum cost Steiner tree G_z.

Because it is not easy to design and verify self-stabilizing algorithms, several techniques have been proposed for constructing self-stabilizing algorithms. One of the techniques is *fair composition* [5]. Let A_1 and A_2 be correct self-stabilizing algorithms. Fair execution of these two algorithms is to execute them in a fair fashion at each process. For example, execution of A_1 and A_2 alternatively (i.e., round robin) at each process is a fair execution. A_2 may refer to variables of A_1 as long as A_1 never refers to ones of A_2. Because A_1 eventually stabilizes, so does A_2, and thus specifications of A_1 and A_2 eventually become true. Thus, fair composition of A_1 and A_2 yields a correct self-stabilizing algorithm A that satisfies the specifications of A_1 and A_2 at the same time. It is easy to extend above method to compose any number of algorithms.

LSS-ST is constructed by fair composition of the following four layers (algorithms).

In the first layer algorithm SS-SPF, we compute a shortest path forest $G_F = (V, E_F, w)$ for underlaying network $G = (V, E, w)$ and $Z \subseteq V$, in which each tree is rooted by a Z-member. This layer corresponds to Step 1 of Chen et al.'s algorithm.

Together with the second and the third layers, we compute the minimum spanning tree where each tree in the shortest path forest is regarded as a node. In other words, we compute the minimum spanning tree G_s of a virtual graph $G_c = (Z, E_c, w_c)$, where $(P^z_i, P^z_j) \in E_c$ iff trees rooted by P^z_i and P^z_j of G_F are neighbors, and w_c is the cost of the shortest path between P^z_i and P^z_j in G_F. To obtain essentially the same result on graph (network) G, we modify the costs of edges of G and obtain the network $G_P = (V, E, w_P)$ in the second layer (algorithm SS-TN), and compute the minimum spanning tree $G_M = (V, E_M, w_P)$ for the network G_P in the third layer. We can use, for example, an algorithm proposed in [1] for the third layer. The second layer corresponds to Step 2, and the third layer corresponds to Step 3 of Chen et al.'s algorithm.

The fourth layer algorithm SS-Pruned prunes the minimum spanning tree $G_M = (V, E_M)$ and obtain an approximate solution $G_z = (V_z, E_z)$. This layer corresponds to Step 4 of Chen et al.'s algorithm.

4.1 The First Layer: Construction of a Shortest Path Forest

First, we propose a self-stabilizing algorithm SS-SPF for finding a shortest path forest $G_F = (V, E_F, w)$ on G.

The input to each process P_i are
(1) $z(P_i)$, a constant which is true iff P_i is in Z, and
(2) $w(P_i)[P_j]$, the cost of incident edge for each $P_j \in N_i$.
The output of each process are
(1) $D(P_i)$, the distance from the nearest Z-member, i.e. the distance of the shortest path between P_i and the Z-member,
(2) $F(P_i)$, the father in the shortest path forest, and
(3) $R(P_i)$, process id of the nearest Z-member.

Formal description of the proposed algorithm SS-SPF is shown in Fig. 2. The outline of SS-SPF is as follows. Each Z-member becomes a root of a tree by GC1. Other process (non-Z-member) executes GC2 to decide its father that yields the smallest distance from a Z-member. The legitimate configuration is defined as follows.

Definition 6: A configuration of SS-SPF is legitimate iff each process $P_i \in V$ satisfies the following three conditions.
- Condition A1: $D(P_i)$ is the distance from the nearest Z-member. If P_i is a Z-member, then $D(P_i) = 0$.
- Condition A2: $F(P_i)$ is the father of P_i in the shortest path tree rooted by the nearest Z-member. If P_i is in the same distance from two or more Z-members, P_i selects a neighbor process with the smallest id among candidates for its father. If P_i is a Z-member, then $F(P_i) = P_i$.
- Condition A3: $R(P_i)$ is the process id of the nearest Z-member. If P_i is in the same distance from two or more Z-members, $R(P_i)$ is the same value with $R(F(P_i))$. If P_i is a Z-member, then $R(P_i) = P_i$. □

By Λ_f, we denote a set of legitimate configurations. We show correctness of this algorithm below.

Lemma 1: No process is privileged in configuration γ if and only if $\gamma \in \Lambda_f$.

Proof: It is clear that no process is privileged if the configuration is legitimate.

We consider configuration γ in which no process is privileged. For any Z-member P_z, $D(P_z) = 0$ and $F(P_z) = R(P_z) = P_z$ hold by definition of GC1, and thus conditions A1, A2 and A3 of Definition 6 hold at P_z. Therefore, we consider non-Z-member bellow.

We show that the value of $D(P_i)$ is equal to the distance

Constant
$z(P_i) \in \{0,1\}$: $z(P_i) = 1$ iff P_i is a Z-member.
$w(P_i)[P_j] \geq 1$: nonnegative cost of the edge between P_i and P_j.
Local Variable
$D(P_i)$: distance from the nearest Z-member.
$F(P_i)$: the father process of P_i in a shortest path forest.
$R(P_i)$: the nearest Z-member process.
Macro
$Distance(P_i) \equiv \min_{P_j \in N_i}(D(P_j) + w(P_i)[P_j])$
$Father(P_i) \equiv \min\{id(P_j) \mid P_j \in N_i, D(P_j) + w(P_i)[P_j] = Distance(P_i)\}$

A Set of Guarded-Commands:
```
*[
  /* GC1: Z-member becomes a root of a shortest path tree.*/
  z(P_i) = 1∧
  {D(P_i) ≠ 0 ∨ F(P_i) ≠ P_i ∨ R(P_i) ≠ P_i}
    → D(P_i) = 0; F(P_i) = P_i; R(P_i) = P_i;

  /* GC2: Non-Z-member selects a neighbor as a father
       and updates the distance.*/
  z(P_i) = 0∧
  {D(P_i) ≠ Distance(P_i) ∨ F(P_i) ≠ Father(P_i) ∨ R(P_i) ≠ R(Father(P_i))}
    → D(P_i) = Distance(P_i); F(P_i) = Father(P_i); R(P_i) = R(Father(P_i));
]
```

Fig. 2 SS-SPF: A self-stabilizing algorithm for constructing a shortest path forest.

from the nearest Z-member process for each P_i.

Suppose that some P_i holds wrong value in $D(P_i)$. Formally, suppose that there exist three processes $P_z \in Z$, $P_i \notin Z$ and $P_j \in N_i$ such that P_z is the nearest Z-member of both P_i and P_j, $\mathrm{dist}(P_i, P_z) = \mathrm{dist}(P_j, P_z) + w(P_i)[P_j]$, $D(P_j) = \mathrm{dist}(P_j, P_z)$ and $D(P_i) \neq \mathrm{dist}(P_i, P_z)$. Then, by definition of the macro $Distance(P_i)$ in Fig. 2, we have

$$\begin{aligned} D(P_i) &= Distance(P_i) \\ &= \min_{P_h \in N_i}\{D(P_h) + w(P_i)[P_h]\} \\ &= D(Father(P_i)) + w(P_i)[Father(P_i)]. \end{aligned}$$

Therefore, P_i recognizes its neighbor $P_h \neq P_j$ as *Father* of P_i. By following *Father* link from P_i, we obtain a sequence $(P_i, P_h, \cdots)$. The sequence cannot be infinite, and let P_k be the terminal process of the sequence.

First, suppose that $P_k (\neq P_z)$ is a Z-member. Then, we have $D(P_k) = \mathrm{dist}(P_k, P_k) = 0$, and by following *Father* in reverse order,

$$\begin{aligned} D(P_h) + w(P_i)[P_h] &> \mathrm{dist}(P_j, P_z) + w(P_i)[P_j] \\ &= D(P_j) + w(P_i)[P_j]. \end{aligned}$$

Therefore, P_i is privileged and this is a contradiction.

Next, we suppose that P_k is not a Z-member. Then P_k must have the minimum value of D. However,

$$\begin{aligned} D(P_k) &\neq Distance(P_k) \\ &= \min_{P_q \in N_k}\{D(P_q) + w(P_k)[P_q]\}. \end{aligned}$$

Therefore, P_k is privileged and this is also a contradiction.

Therefore, for each P_i, $D(P_i) = \mathrm{dist}(P_i, P_z)$ holds, where P_z is the nearest Z-member of P_i, in configuration γ.

Finally, we consider other variables. Each non-Z-member process P_i holds the following values by GC2.

- $D(P_i) = Distance(P_i) = \min_{P_j \in N_i}\{D(P_j) + w(P_i)[P_j]\} = D(Father(P_i)) + w(P_i)[Father(P_i)]$,
- $F(P_i) = Father(P_i)$ and
- $R(P_i) = R(Father(P_i))$

If $\mathrm{dist}(P_i, P'_z) = \mathrm{dist}(P_i, P''_z)$, for some Z-members P'_z and P''_z, then a neighbor process with smaller process id is selected by macro $Father(P_i)$. Because the value of $Father(P_i)$ is the father of P_i in a shortest path tree for each P_i, trees are disjoint and each tree is rooted by a Z-member. Thus, the configuration satisfies conditions A1, A2 and A3.

Therefore, the configuration is legitimate when no process is privileged. □

Lemma 2: For any configuration γ_0 and any computation starting from γ_0, eventually no process is privileged.

Proof: First, observe that guarded command that each Z-member process P_i (i.e., $z(P_i) = 1$) can execute is only GC1. By definition of GC1, each Z-member process executes GC1 at most once. Other processes can execute GC2.

Now, suppose that there exists an infinite (non-converging) computation starting from γ_0. Let K be a set of processes that are executed infinitely often, and let $J = V - K$. Without loss of generality, we can assume that only processes in K are executed in the infinite computation. Because each Z-member can be executed at most once, the set K does not include any Z-members. Let $K' = \cup_{P_j \in J} N_j \cap K$ be a set of neighbor processes of J in K. Let $\gamma_0, \gamma_1, \gamma_2, \ldots$ be a sequence of the infinite computation. Let $D_{min}(\gamma_t)$ be the minimum of $D(P_i)$ among $P_i \in K$ in configuration γ_t. We simply write D_{min} when γ_t is clear from context.

If every process P_i with $D(P_i) = D_{min}$ executes GC2, then D_{min} must increase eventually, because the value D of the neighbors of P_i is larger than D_{min} and P_i cannot execute GC2 infinitely often without changing the value D. Because D_{min} eventually becomes larger than any $D(P_j)$ for any $P_j \in J$, the value of $D(P_i)$ for each $P_i \in K$ becomes larger than any value of $D(P_j)$ for any $P_j \in J$. Thus some process $P_i \in K'$ eventually selects a process in J as its father. When P_i selects a process in J as its father, because each process in J never executes, P_i is not executed forever. □

Theorem 1: The algorithm SS-SPF is self-stabilizing with respect to Λ_f.

Proof: By Lemma 2, there is no infinite computation for any initial configuration. By Lemma 1, any terminal configuration is a configuration in which the shortest path forest is computed. Thus, the theorem holds. □

Because the convergence time depends on the maximal value of the costs of edges, we assume here that the maximal value of the cost of each edge is M. The maximum value of D must be smaller than $(n-1)M$. Let the maximum value of D be $D_{\max} = (n-1)M$. We assume that if $Distance(P_i)$ becomes larger than $D_{\max}$, P_i never increases the value of $D(P_i)$. In other words, we add the condition $Distance(P_i) \le D_{\max}$ to the guard of GC2.

Theorem 2: Time complexity of SS-SPF is $O(n^4)$.

Proof: By GC1 and 2, each process creates a shortest path forest. During convergence process, an incorrect tree may be temporarily created with a root process with incorrect D (distance) value. If a Z-member is executed once, it holds correct distance value in D and remains so forever. Thus, the maximum number of convergence steps is obtained by a computation in which executions of Z-members and some non-Z-members with incorrect D value are avoided if possible. Let X be a set that includes all Z-members and some non-Z-members with incorrect D value.

First, we consider a computation in which the members of X do not execute any guarded commands. In this computation, a shortest path tree may be created with a non-Z-member, say P_0, such that P_0 is not a member of X and P_0 holds the smallest value of D among $V - X$.

We claim that construction of such a tree requires $O(n^2)$ steps. Let Q_h be a set of processes P_k such that a shortest path from P_0 to P_k has h edges, and H be the maximum h such that $Q_h \neq \emptyset$. First, execute processes in Q_H. Next, execute processes in Q_{H-1}, which may make some processes in Q_H privileged again. Then, execute processes in Q_H. Similarly, we iterate execution of processes in $Q_h, Q_{h+1}, \ldots, Q_H$, for each $h = H, H-1, \cdots, 1$. Since $\Sigma|Q_h| \le n$, it is not difficult to see that the number of execution is at most n^2.

The smallest value of D among $V - X$ is $0 < D \le D_{max}$, and the smallest value of D increases at least by one because we assume that each cost is at least one. Therefore, incorrect trees are created at most D_{max} times. Then, the members of X must execute. Since the number of the member of X is at most n, we obtain a bound of the maximum convergence $n^2 \cdot D_{max} \cdot n = n^3 \cdot D_{max}$. Because the value of $D_{\max}$ is bounded by $(n-1)M$, we have $n^3(n-1)M = O(n^4)$. □

4.2 The Second Layer: Modification of Edge Cost

The second layer algorithm SS-TN computes a network $G_P = (V, E, w_P)$ from G_F, where

$$w_P(e) = \begin{cases} 0 & \text{if } e \text{ is a tree edge,} \\ \infty & \text{if } e \text{ is an intra-tree edge,} \\ \mathrm{dist}(P_i, P_z) + w(P_i, P_j) + \mathrm{dist}(P_j, P'_z) & \\ \quad \text{if } e = (P_i, P_j) \text{ is an inter-tree edge,} & \\ \quad \text{and } P_z \text{ (resp. } P'_z\text{) is the root of } P_i & \\ \quad \text{(resp. } P_j\text{).} & \end{cases}$$

The input to each process P_i of SS-TN is

(1) $w(P_i)[P_j]$, the costs of incident edges, where P_j is a neighbor of P_i,
(2) $D(P_i)$, the distance from the nearest Z-member,
(3) $F(P_i)$, the father of P_i in a shortest path forest, and
(4) $R(P_i)$, process id of the nearest Z-member.

The output of each process P_i of SS-TN is the cost $W(P_i)[P_j]$ of each edge between P_i and P_j, where P_j is a neighbor of P_i. A collection of W forms a cost function w_P of G_P.

Constant
$w(P_i)[P_j]$: cost of the edge between P_i and P_j.
$D(P_i)$: distance from Z-member.
$F(P_i)$: the father of P_i on the shortest path forest.
$R(P_i)$: the process id of the nearest Z-member.
Local Variable
$W(P_i)[P_j]$: new cost of the edge between P_i and P_j.

A Set of Guarded-Commands:
```
*[
  /* GC1: Tree edge */
  ∃P_j ∈ N_i[R(P_i) = R(P_j) ∧ (P_j = F(P_i) ∨ P_i = F(P_j)) ∧ W(P_i)[P_j] ≠ 0]
     → W(P_i)[P_j] = 0;

  /* GC2: Intra-tree edge */
  ∃P_j ∈ N_i[R(P_i) = R(P_j) ∧ ¬(P_j = F(P_i) ∨ P_i = F(P_j)) ∧ W(P_i)[P_j] ≠ ∞]
     → W(P_i)[P_j] = ∞;

  /* GC3: Inter-tree edge */
  ∃P_j ∈ N_i[R(P_i) ≠ R(P_j) ∧ W(P_i)[P_j] ≠ w(P_i)[P_j] + D(P_i) + D(P_j)]
     → W(P_i)[P_j] = w(P_i)[P_j] + D(P_i) + D(P_j);
]
```

Fig. 3 SS-TN: A self-stabilizing algorithm for transforming the network.

The outline of SS-TN is as follows. Each process classifies each of its incident edges. For a tree (resp. intra-tree, inter-tree) edge e, P_i computes the cost of e by GC1 (resp. GC2, GC3). Formal description of the proposed algorithm SS-TN is shown in Fig. 3, and the legitimate configuration is defined as follows.

Definition 7: A configuration of SS-TN is legitimate iff each edge $e = (P_i, P_j) \in E$ satisfies the following three conditions.

- Condition B1: If e is a tree edge, then $W(P_i)[P_j]$ is 0,
- Condition B2: If e is an intra-tree edge, then $W(P_i)[P_j]$ is ∞, and
- Condition B3: If e is an inter-tree edge, then $W(P_i)[P_j]$ is the distance of the shortest path via e between $R(P_i)$ and $R(P_j)$. □

By Λ_t, we denote a set of legitimate configurations.

Lemma 3: No process is privileged in configuration γ if and only if $\gamma \in \Lambda_t$.

Proof: Consider any non-legitimate configuration.

- Suppose that condition B1 is false. Then, there exists a tree edge $e = (P_i, P_j)$ such that $W(P_i)[P_j] \neq 0$, $R(P_i) = R(P_j)$, and $P_j = F(P_i) \vee P_i = F(P_j)$. Thus the guard of GC1 is true at P_i and P_j.
- Suppose that condition B2 is false. Then, there exists an intra-tree edge $e = (P_i, P_j)$ such that $W(P_i)[P_j] \neq \infty$, $R(P_i) = R(P_j)$, and $P_j \neq F(P_i) \wedge P_i \neq F(P_j)$. Thus, the guard of GC2 is true at P_i and P_j.
- Suppose that the condition B3 is false. Then, there exists an inter-tree edge $e = (P_i, P_j)$ such that $W(P_i)[P_j]$ which is not distance of the shortest path via e between $R(P_i)$ and $R(P_j)$, and $R(P_i) \neq R(P_j)$. Thus, the guard of GC3 is true at P_i and P_j.

Therefore, at least one process is privileged in non-legitimate configuration.

It is clear that no process is privileged if the configuration is legitimate. □

Lemma 4: For any configuration γ_0 and any computation starting from γ_0, eventually no process is privileged.

Proof: Let $e = (P_i, P_j)$ be any edge. P_i executes a guarded command for e at most once. Thus, there is no infinite computation. □

By Lemma 3 and Lemma 4, we have the following theorem.

Theorem 3: The algorithm SS-TN is self-stabilizing with respect to Λ_t. □

Theorem 4: Time complexity of SS-TN is $O(n^2)$.

Proof: Each process executes guarded command at most once for each incident edge. Therefore, each process P_i executes at most Δ_i, where Δ_i is the degree of P_i. Time complexity of this algorithm is bounded by $\Sigma_i \Delta_i = 2|E| = O(n^2)$. □

4.3 The Fourth Layer: Pruned-MST

This layer (algorithm SS-Pruned) prunes $G_M = (V, E_M)$ computed by the third layer. We assume that a minimum spanning tree G_M is rooted. A unrooted minimum spanning tree computed in the third layer is modified to a rooted one by the following procedure. A process P_i such that $\mathrm{src}_i = 1$ becomes the root, each process computes the distance from the root. Each process decides its father and children based on the distance of itself and its neighbors. Because this technique is widely used (e.g., [3]), we omit formal description and its proof of correctness.

The input to each process P_i of SS-Pruned is

(1) $z(P_i)$, a constant which is true iff P_i is in Z, and
(2) $CH(P_i) \subseteq N_i$, a set of children in a minimum spanning tree G_M.

The output of each process P_i is a boolean $S(P_i)$ if P_i is a member of the Steiner tree (G_z). A subgraph of G induced by a set of processes P_i such that $S(P_i)$ is true is the solution of LSS-ST.

Formal description of proposed algorithm SS-Pruned is shown in Fig. 4. There are four guarded commands (GC1-GC4) in the algorithm. A Z-member joins the Steiner tree by GC1. A leaf of G_M which is not a Z-member must not join the Steiner tree by GC2. Other processes may execute GC3 and GC4. A process must not be a member of the Steiner tree by GC3 if all children are not member of the Steiner tree. Otherwise, a process must join the Steiner tree by GC4. The legitimate configuration is defined as follows.

Definition 8: For any configuration γ of SS-Pruned, on a minimum spanning tree $G_M = (V, E_M)$, consider a subset $V_z = \{P_i \mid S(P_i) = 1\}$ of V. A configuration γ is legitimate iff a subgraph $G_z = (V_z, E_M \cap (V_z \times V_z))$ of G satisfies the following three conditions.

Constant
$z(P_i) \in \{0,1\}$: $z(P_i) = 1$ if P_i is Z-member.
$CH(P_i)$: a set of children of P_i in G_M.
Local Variable
$S(P_i) \in \{0,1\}$ —$S(P_i) = 1$ (resp. 0) if P_i is a Steiner tree's member (resp. non-member).

A Set of Guarded-Commands:

```
*[
 /* GC1: Z-member joins a Steiner tree.*/
 S(P_i) = 0 ∧ z(P_i) = 1
   → S(P_i) := 1;

 /* GC2: Non-Z-member leaf of G_M resigns a Steiner tree.*/
 S(P_i) = 1 ∧ z(P_i) = 0 ∧ CH(P_i) = NULL
   → S(P_i) := 0;

 /* GC3: If every children resign Steiner tree, resign the tree.*/
 S(P_i) = 1 ∧ z(P_i) = 0 ∧ ∀P_j ∈ CH(P_i)[S(P_j) = 0]
   → S(P_i) := 0;

 /* GC4: If at least one Steiner tree's member exist in the children,
     join the tree.*/
 S(P_i) = 0 ∧ z(P_i) = 0 ∧ ∃P_j ∈ CH(P_i)[S(P_j) = 1]
   → S(P_i) := 1;
]
```

Fig. 4 SS-Pruned: A self-stabilizing algorithm for pruning a minimum spanning tree.

- Condition C1: G_z is a tree and connected. (This implies that the father P_j of process P_i with $S(P_i) = 1$ must have $S(P_j) = 1$.)
- Condition C2: For each leaf P_i of G_z, $z(P_i) = 1$ must hold. (This implies that if all descendants of P_i have $S = 0$ and P_i is not a Z-member, then $S(P_i) = 0$ must hold.)
- Condition C3: A set $\{P_i \mid z(P_i) = 1\}$ must be a subset or equal to V_z. □

By Λ_p, we denote a set of legitimate configurations.

Lemma 5: At least one process is privileged in configuration γ if and only if $\gamma \notin \Lambda_p$.

Proof: Consider any non-legitimate configuration γ. Then, there are three cases to consider.

- Suppose that condition C1 is false. In γ, there exists a process P_i such that $S(P_i) = 1$ and $S(P_j) = 0$, where P_j is the father of P_i. Then the guard of GC4 is true at P_j.
- Suppose that condition C2 is false. In γ, there exists a process P_i such that P_i is a leaf of G_z and $z(P_i) = 0$. Then the guard of GC3 is true at P_i.
- Suppose that condition C3 is false. In γ, there exists a process P_i such that $S(P_i) = 0$ and $z(P_i) = 1$. Then the guard of GC1 is true at P_i.

Therefore, at least one process is privileged in γ.

Next, suppose that at least one process is privileged in configuration γ. Let P_i be any privileged process in γ.

- Suppose that P_i is privileged by GC1. In this case, $\{P_i \mid z(P_i) = 1\} \nsubseteq V_z$ holds, which implies that condition C3 does not hold.
- Suppose that P_i is privileged by GC2. In this case, P_i must be a leaf of G_M and non-Z-member. Thus, condition C2 does not hold.
- Suppose that P_i is privileged by GC3. In this case, P_i holds $S(P_i) = 1$ and $z(P_i) = 0$ and all the children hold $S = 0$, and thus condition C2 does not hold.
- Suppose that P_i is privileged by GC4. In this case, P_i holds $S(P_i) = 0$ and $z(P_i) = 0$ and at least one of the children P_j holds $S(P_j) = 1$, and thus condition C1 does not hold.

Thus, if there exists a privileged process in γ, then γ is not legitimate. □

Lemma 6: For any configuration γ_0 and any computation starting from γ_0, eventually no process is privileged.

Proof: First, consider any Z-member process P_i such that $z(P_i) = 1$. Guarded command that it may execute is only GC1. By definition of GC1, it executes GC1 at most once. Consider any leaf P_j of G_M which is non-Z-member i.e., $z(P_i) = 0$. It never executes GC1. Because it does not have children, it never executes GC3 and GC4. By definition of GC2, it can execute GC2 at most once.

Now, suppose that there exists an infinite computation starting from γ_0. Then, there must be at least one process, say P_i, which executes infinitely often. It must be neither a Z-member nor a leaf of G_M. By definition of GC3 and GC4, P_i alternates executing guarded commands GC3 and GC4 forever. Note that after P_i executes GC3 or GC4, at least one of the children of P_i must change its state. Therefore, there exists a process $P_k \in CH(P_i)$ such that P_k is executed infinitely often. By repeating this discussion, there exists a leaf process of G_M which is a descendant of P_i which is executed infinitely often, which is impossible.

Therefore, there is no infinite computation, and eventually computation terminates in a configuration in which no process is privileged. □

Theorem 5: The algorithm SS-Pruned is self-stabilizing with respect to Λ_p.

Proof: By Lemma 6, there is no infinite computation for any initial configuration. By Lemma 5, in any terminal configuration, a Steiner tree is computed. Thus, the theorem holds. □

Theorem 6: Time complexity of SS-Pruned is $O(n^2)$.

Proof: GC1 is executed by Z-members. Z-members is $|Z|$, and each of them executes at most once. So, the number of steps by them is at most $|Z| \le n$.

G_M's leaf except Z-member execute GC2, and the number of such processes is at most $n - |Z|$. Since, each of them executes at most once, the number of steps by them is at most $n - |Z|$.

Other processes execute GC3 and GC4, and the number of such processes is at most $n - |Z|$. We say that a process P_i holds a "token" if and only if P_i is privileged by GC3 or GC4. The number of tokens in configuration is at most $n - |Z|$. By execution of GC3 or GC4 at P_i with a token, the token at P_i moves to its father, or it disappears (if it arrives at the Z-members). Because the number of edges is at most $n-1$, each token disappears within $n-1$ executions of GC3 and GC4. Thus, the number of executions of GC3 and GC4 is at most $(n-1)(n-|Z|)$.

The total number of execution steps is $n^2 - n|Z| + |Z|$. Therefore, the time complexity is $O(n^2)$. □

5. Conclusion

In this paper, we proposed a self-stabilizing heuristic algorithm to compute a minimal cost Steiner tree. Our algorithm LSS-ST is constructed by fair composition of four algorithms and based on a heuristic with a generalized minimum spanning tree.

We analyze time complexity of each layer. They look like not good results, but they are assumed unfair scheduler.

The competitiveness is $2(1-1/l)$ where l is the number of the leaves of the optimal Steiner tree, and competitiveness is bounded by 2. Thus, competitiveness of our algorithm is better than the simple heuristic by Pruned-MST whose competitiveness is $n-|Z|+1$. However, there exists sequential heuristic algorithms whose competitiveness is 1.55. Therefore, development of a self-stabilizing distributed algorithm with competitiveness 1.55 is left as a future task.

References

[1] G. Antonoiu and P.K. Srimani, "Distributed self-stabilizing algorithm for minimum spanning tree construction," Euro-Par'97 Parallel Proceeding, LNCS, vol.1300, pp.480–487, 1997.
[2] G.-H. Chen, M.E. Houle, and M.-T. Kuo, "The Steiner problem in distributed computing systems," Inf. Sci., vol.74, pp.73–96, 1993.
[3] N.-S. Chen, H.-P. Yu, and S.-T. Huang, "A self-stabilizing algorithm for constructing spanning trees," Inf. Process. Lett., vol.39, pp.147–151, 1991.
[4] E.W. Dijkstra, "Self-stabilizing systems in spite of distributed control," Commun. ACM, vol.17, no.11, pp.643–644, 1974.
[5] S. Dolev, Self-Stabilization, The MIT Press, 2000.
[6] F.C. Gärtner, "Fundamentals of fault-tolerant distributed computing in asynchronous environments," ACM Comput. Surveys, vol.31, no.1, pp.1–26, 1996.
[7] C. Gropl, S. Hougardy, T. Nierhoff, and H.J. Promel, "Approximation algorithms for the Steiner tree problem in graphs," in Steiner Trees in Industry, ed. X. Cheng and D.-Z. Du, pp.235–279, Kluwer Academic Publishers, 2001.
[8] R.M. Karp, Reducibility among combinatorial problems, Plenum Press, New York, 1972.
[9] L. Kou, G. Markowsky, and L. Berman, "A fast algorithm for Steiner trees," Acta Informatica, vol.15, pp.141–145, 1981.
[10] K. Mehlhorn, "A faster applozimation algorithm for the Steiner problem in graphs," Inf. Process. Lett., vol.27, pp.125–128, 1988.
[11] V.J. Rayward-Smith, "The computation of nearly-minimal Steiner trees in graphs," Int. J. Math. Educ. Sci. Technol., vol.14, pp.15–23, 1983.
[12] G. Robins and A. Zelikovsky, "Improved Steiner tree approximation in graphs," Proc. Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2000, pp.770–779, 2000.
[13] H. Takahashi and A. Matsuyama, "An approximate solution for the Steiner problem in graphs," Math. Japonica, vol.24, pp.573–577, 1980.
[14] D. Waitzman, C. Partridge, and S. Deering, "Distance vector multicast routing protocol," RFC:1075, Nov. 1988.
[15] P. Winter, "Steiner problem in networks: A survey," Networks, vol.17, pp.129–167, 1987.
[16] Y.F. Wu, P. Widmayer, and C.K. Wong, "A faster approximation algorithm for the Steiner problem in graphs," Acta Informatica, vol.23, pp.223–229, 1986.

Sayaka Kamei received the B.E. and M.E. degrees in electronics engineering in 2001, 2003 respectively from Hiroshima University. She is currently a doctor course student of Hiroshima University. Her research interests include distributed algorithms.

Hirotsugu Kakugawa received the B.E. degree in electronics engineering in 1990 from Yamaguchi University, and the M.E. and D.E. degrees in information engineering in 1992, 1995 respectively from Hiroshima University. He is currently an associate professor of Hiroshima University. His research interests include distributed algorithms and computer typography. He is a member of the IEEE Computer Society and the Information Processing Society of Japan.