A Self-Stabilizing 3-Approximation for the Maximum Leaf Spanning Tree Problem in Arbitrary Networks

Sayaka Kamei[1⋆], Hirotsugu Kakugawa[2⋆⋆], Stéphane Devismes[3], and Sébastien Tixeuil[4⋆⋆⋆]

[1] Dept. of Information Engineering, Hiroshima University, Japan,
E-mail: s-kamei@se.hiroshima-u.ac.jp
[2] Dept. of Computer Science, Osaka University, Japan,
E-mail: kakugawa@ist.osaka-u.ac.jp
[3] Université Joseph Fourier, Grenoble I, France, E-mail: Stephane.Devismes@imag.fr
[4] LIP6 UMR 7606, Université Pierre et Marie Curie, France,
E-mail: Sebastien.Tixeuil@lip6.fr

Abstract. The maximum leaf spanning tree (MLST) is a good candidate for constructing a virtual backbone in self-organized multihop wireless networks, but is practically intractable (NP-complete). Self-stabilization is a general technique that permits to recover from catastrophic transient failures in self-organized networks without human intervention. We propose a fully distributed self-stabilizing approximation algorithm for the MLST problem on arbitrary topology networks. Our algorithm is the first self-stabilizing protocol that is specifically designed for the construction of an MLST. It improves other previous self-stabilizing solutions both for generality (arbitrary topology graphs *vs.* unit disk graphs or generalized unit disk graphs, respectively) and for approximation ratio, as it guarantees the number of its leaves is at least $1/3$ of the maximum one. The time complexity of our algorithm is $O(n^2)$ rounds.

1 Introduction

Multihop wireless ad hoc or sensor networks have neither fixed physical infrastructure nor central administration. They typically operate in a self-organizing manner that permits them to autonomously construct routing and communication primitives that are used by higher level applications. The construction of virtual backbones infrastructures usually makes use of graph related properties over the graph induced by communication capabilities (*i.e.* nodes represent machines, and edges represent the ability for two machines within wireless range to communicate) of the network. For example, a connected dominating set (CDS) is a good candidate for a virtual backbone since it guarantees reachability of every node yet preserves energy. The maximum leaf spanning tree (MLST) problem consists in constructing a spanning tree with the maximum number of leaves. Finding the MLST is tantamount to finding the minimum CDS: let $G = (V, E)$

⋆ This work is supported in part by a Grant-in-Aid for Young Scientists ((B)22700074) of JSPS.
⋆⋆ This work is supported in part by Kayamori Foundation of Informational Science Advancement, a Grant-in-Aid for Scientific Research ((B)20300012) of JSPS, and "Global COE (Centers of Excellence) Program" of the Ministry of Education, Culture, Sports, Science and Technology, Japan.
⋆⋆⋆ This work is supported in part by ANR projects SHAMAN and ALADDIN

be a graph and $cds(G)$ be the size of the minimum CDS of G, then $|V| - cds(G)$ is the number of leaves of the MLST of G [1].

One of the most versatile techniques to ensure forward recovery of distributed systems and networks is that of *self-stabilization* [2]. A distributed algorithm is self-stabilizing if after faults and attacks hit the system and place it in some arbitrary global state, the system recovers from this catastrophic situation without external (*e.g.* human) intervention in finite time. As self-stabilization makes no hypothesis about the nature or the extent of the faults (self-stabilization only deals with the effect of the faults), it can also be used to deal with other transient changes while the network is being operated (topology change, message loss, spontaneous resets, etc.).

1.1 Related Works

In [3], Galbiati *et al.* proved that the MLST problem is MAX-SNP-hard, *i.e.*, there exists $\varepsilon > 0$ such that finding approximation algorithm[5] with approximation ratio $1+\varepsilon$ is NP-hard. In [1], Solis-Oba proposed a 2-approximation algorithm, and in [4], Lu *et al.* proposed a 3-approximation algorithm. Note that none of those algorithms [1, 4] is distributed, not to mention self-stabilizing.

Spanning tree construction is one of the main studied problems in self-stabilizing literature. One of the main recent trends in this topic is to provide self-stabilizing protocols for constrained variants of the spanning tree problem, *e.g.* [5], [6], [7], etc. None of those metrics give any guarantee on the number of leaves.

In [8], Guha *et al.* showed that the existence of an algorithm for finding the minimum CDS with approximation ratio α implies the existence of an algorithm of the MLST problem with approximation ratio 2α. In turn, there exist self-stabilizing approximation algorithms for finding the minimum CDS. In [9], Kamei *et al.* proposed a self-stabilizing 7.6-approximation algorithm for the CDS problem in unit disk graphs, *i.e.*, this algorithm is also an approximation algorithm for the MLST problem with approximation ratio 15.2 (by [8]). However, this algorithm [9] does not guarantee any approximation ratio in general topology networks. The subsequent work of Raei *et al.* [10] proposed a self-stabilizing $20\lfloor \ln R / \ln(2\cos(\pi/5))\rfloor$-approximation algorithm in generalized disk graphs where $R = r_{max}/r_{min}$ and r_{max} (resp. r_{min}) is the maximum (resp. minimum) transmission range. This algorithm is an approximation for the MLST problem with approximation ratio $40\lfloor \ln R / \ln(2\cos(\pi/5))\rfloor$. Again, this algorithm [10] does not guarantee any approximation ratio in general topology networks.

1.2 Our contribution and outline of this paper

We propose a fully distributed self-stabilizing approximation algorithm for the MLST problem on arbitrary topology networks. Its time complexity is $O(n^2)$ rounds. To our knowledge, our algorithm is the first self-stabilizing protocol that is specifically designed for the construction of MLST. It improves over previous self-stabilizing derivations both for generality (arbitrary topology graphs *vs.* unit disk graphs [9] (resp. gen-

[5] An approximation algorithm for the MLST problem is an algorithm that guarantees approximation ratio $|T_{opt}|/|T_{alg}|$, where $|T_{alg}|$ is the number of leaves obtained by the approximation algorithm in the worst case and $|T_{opt}|$ is the number of leaves of the optimal solution.

eralized disk graphs [10])) and for approximation ratio (3 *vs.* 15.2 [9] (resp. $40\lfloor \ln R / \ln(2\cos(\pi/5)) \rfloor$ [10])).

The improved approximation ratio permits to improve significantly the load and the energy consumed by the virtual backbone. The improved generality on the communication graph enables our scheme to be useful even in networks that cannot be modeled by (generalized) disk graphs (such as wired networks).

This paper is organized as follows. In Section 2, we formally describe the system model and the distributed MLST problem. In Section 3, we present our self-stabilizing approximation algorithm for the distributed MLST problem, and prove the correctness and analyze the time complexity of the proposed algorithm. Details of proofs are omitted because of limitation of space, and they will appear in the full paper. Concluding remarks can be found in Section 4.

2 Preliminaries

Let $V = \{P_1, P_2, ..., P_n\}$ be a set of n processes and $E \subseteq V \times V$ be a set of bidirectional communication links in a distributed system. Each links is an unordered pair of distinct processes. Then, the topology of the distributed system is represented as an undirected graph $G = (V, E)$. We assume that G is connected and simple. In this paper, we use "graphs" and "distributed systems" interchangeably. We assume that each process has unique identifier. By P_i, we denote the identifier of process P_i for each process P_i.

We call *subgraph* of G any graph $G' = (V', E')$ such that $V' \subseteq V$, $E' \subseteq E$, and $\forall P_i, P_j$, $(P_i, P_j) \in E' \Rightarrow P_i, P_j \in V'$. By N_i, we denote the set of neighboring processes of P_i. For each process P_i, the set N_i is assumed to be a constant. We define the *degree* of P_i as the number of its neighbors. The degree of P_i in the subgraph G' is the number of edges of E' incident to P_i. We assume that the maximum degree of G is at least 3. We define the *distance* between P_i and P_j as the number of the edges of the shortest path between them.

As communication model, we assume that each process can read the local state of neighboring processes. This model is called the *state reading model*. Although a process can read the local state of neighboring processes, it cannot update them; it can only update its local state.

A set of local variables defines the local state of a process. By Q_i, we denote the local state of each process $P_i \in V$. A tuple of the 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.

We say that P_i is *privileged* in a configuration γ if and only if at least one of the conditions of the algorithm is true and P_i must change the value of its variables in γ. An atomic step of each process P_i consists of following three sub-steps: (1) read the local states of all neighbors and evaluate the conditions of the algorithm, (2) compute the next local state, and (3) update the local state.

Executions of processes are scheduled by an external (virtual) scheduler called *daemon*. That is, the daemon decides which processes to execute in the next step. Here, we assume a *distributed weakly fair daemon*. *Distributed* means that, at each step, the daemon selects an arbitrary non-empty set of privileged processes, and selected processes executes the atomic step in parallel. *Weakly fair* means that every continuously privileged process will be eventually executed.

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

Definition 1 (Self-Stabilization). *Let Γ be a 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 as long as the system does not fail.*

Each $\gamma \in \Lambda$ is called a legitimate *configuration. Conversely, any configuration that is not legitimate is said* illegitimate. □

A *spanning tree* $T = (V', E')$ is any acyclic connected subgraph of G such that $V' = V$ and $E' \subseteq E$. A *leaf* of a spanning tree is any process of degree one. Generally, the MLST problem is defined as follows.

Definition 2. *The* maximum leaf spanning tree *is a spanning tree whose number of leaves is maximum.* □

We consider solving the MLST problem in distributed systems in this paper. We assume that each process does not know global information of the network. Under this assumption, we defined the distributed MLST problem as follows.

Definition 3. *Let $G = (V, E)$ be a graph that represents a distributed system. Then, the* distributed maximum leaf spanning tree problem *is defined as follows.*

- *Each process P_i must select a neighbor on G or itself (if the father of P_i is P_i, then P_i is a* root*) as its father on a spanning tree T_{ml} and output it, and*
- *The spanning tree T_{ml} is a maximum leaf spanning tree of G.* □

3 Proposed Algorithm

Our algorithm SSMLST is based on the sequential approximation algorithm in [4].

We call *tree* any subgraph T of G that has no cycle and more than one process. We construct disjoint trees $T_1, T_2, \cdots$, where $T_i = (V_i, E_i)$, $V = V_1 \cup V_2 \cup \cdots$, $|V_i| > 1$, and $V_i \cap V_j = \emptyset$ for any i and j. We call *forest* any set of trees $\{T_1, T_2, \cdots\}$. Note that some process P_i can be alone and does not join the forest, *i.e.*, $S_i = (\{P_i\}, \emptyset)$, in this case P_i is called *singleton*.

Let $d_k(G)$ be the set of nodes that have degree k on G, and let $\bar{d}_k(G)$ be the set of nodes that have degree at least k on G.

Definition 4. *([4]) Let T be a tree of G. If $\bar{d}_3(T)$ is not empty and every node in $d_2(T)$ is adjacent in T to exactly two nodes in $d_3(T)$, then let T be* leafy *tree. Let $T_1, T_2, \cdots$ be disjoint trees on G. If each $T_1, T_2, \cdots$ is leafy, then $F = \{T_1, T_2, \cdots\}$ is a* leafy forest*. If F is not a subgraph of any other leafy forest of G, then we call F* maximal leafy forest. □

In [4], Lu *et al.* showed the following theorem.

Theorem 1. *([4]) Let F be a maximal leafy forest of G, and let T_{ml} be a spanning tree of G such that F is a subgraph of T_{ml}. Let T_{span} be any spanning tree of G. Then, $|d_1(T_{ml})| \geq |d_1(T_{span})|/3$.*

Our algorithm SSMLST first constructs a maximal leafy forest (MLF) of G, and then, it constructs a spanning tree T_{ml} of G that is a supergraph of the MLF. Hence, T_{ml} is an approximation of the MLST with ratio 3 according to Theorem 1.

The proposed algorithm is a fair composition [11] of four layers:

1. In the first layer, each process P_i computes its degree D_i on G and the maximum couple $MAX = (D_0, P_0)$ of degree D_0 and ID P_0 on G, where $(D_i, P_i) > (D_j, P_j) \equiv [D_i > D_j \vee (D_i = D_j \wedge P_i > P_j)]$ for each process P_i and P_j. For this layer, we can use a self-stabilizing leader election algorithm for arbitrary networks, for example [12] (The time complexity of [12] is $O(n)$ rounds.). In such an algorithm, the process with the minimum or the maximum ID is elected as a leader. It is modified to elect the process with the maximum value (D_0, P_0) for our purpose.
2. The second layer SSMLF (see subsection 3.1) computes an MLF on G.
3. The third layer SSTN (see subsection 3.2) modifies the cost of each link based on the MLF.
4. The last layer computes a minimum cost spanning tree T_{ml} based on the costs computed by SSTN. Such costs make T_{ml} includes the MLF. For this layer, we can use one of existing self-stabilizing algorithms, *e.g.*, [6] (The time complexity of [6] is $O(n^2)$ rounds.).

3.1 The Second Layer: Construction of the Maximal Leafy Forest

We now propose a self-stabilizing algorithm called SSMLF that constructs of a maximal leafy forest (MLF) of G. The formal description of SSMLF is shown in Fig. 1.

Each process P_i computes the following outputs:

- $root_i$ is set to $\emptyset$ if P_i is a singleton. Otherwise, P_i belongs to some tree T and $root_i$ is set to the couple (D_r, P_r), where P_r is the root of the tree of P_i.
- $father_i$ is set to the couple (D_j, P_j) where P_j is the father of P_i. If P_i is neither a root nor a singleton, then $P_j \in N_i$. In this case, we say that "P_j *is a father of* P_i" and "P_i *is a child of* P_j". In either cases (P_i is a singleton or the root of its tree), $P_j = P_i$. Note that, in SSMLF, each process P_i distinguishes its incident link to $father_i$ as its parent-link in its tree.
- $rank_i$ is the distance from P_i to the root of its tree.
- $MaxChildren_i$: the *expected number of children* of P_i in its tree. (We shall explain that later.)

In the following explanations, we call *large tree* any tree rooted at a process with a large couple of degree on G and ID. Also, we call a *child-candidate* of process P_i the neighbor of P_i that may become a child of P_i in the future, *e.g.*, singleton, process belonging to a tree that is not larger than the tree of P_i, or process belonging to the tree

Constant (Input)
N_i: the set of neighbors of P_i on G.
D_i: the degree of P_i on G (an output from the first layer).
MAX: the maximum couple of degree and ID on G (an output from the first layer).
Variable (Output)
$root_i = (D_r, P_r)(\emptyset \leq root_i \leq MAX)$: P_r is the root of tree T to which P_i belongs.
$father_i$: the couple (D_j, P_j) of the father P_j of P_i on T.
$rank_i$: the distance from the root to P_i on T.
$MaxChildren_i$: the number of children and child-candidates on T.
Macro

$$MaxRoot_i = \max\{root_j, (D_i, P_i) \mid P_j \in N_i\}$$

$$MinRank_i = \begin{cases} -1 & \text{(In case that } MaxRoot_i = (D_i, P_i)) \\ \min\{rank_j \mid P_j \in N_i \wedge root_j = MaxRoot_i\} & \text{(otherwise)} \end{cases}$$

$$CCand_i = \{P_j \in N_i \mid root_j = \emptyset \vee root_j < MaxRoot_i \vee (root_j = MaxRoot_i \wedge rank_j > MinRank_i + 2)\}$$

$$CountMaxChildren_i = \begin{cases} D_i & \text{(In case that } (D_i, P_i) = MAX) \\ |\{P_j \in N_i \mid father_j = (D_i, P_i)\}| + |CCand_i| & \text{(otherwise)} \end{cases}$$

$$FCand_i = \{P_j \in N_i \mid rank_j + 1 \leq n \wedge (MaxChildren_j \geq 3 \vee (MaxChildren_j = 2 \wedge father_j \neq (D_j, P_j) \wedge root_j > (D_j, P_j)))\}$$

Algorithm for process P_i:
do *forever*{
1 if $(root_i > MAX)$ {
2 $root_i := \emptyset$;
3 } elseif $(MaxChildren_i \neq CountMaxChildren_i)$ {
4 $MaxChildren_i := CountMaxChildren_i$;
/* For the roots. */
5 } elseif $(MaxChildren_i \geq 3 \wedge MaxRoot_i = (D_i, P_i))$ {
6 $root_i := (D_i, P_i)$; $rank_i := 0$; $father_i := (D_i, P_i)$;
/* For other nodes in tree.*/
7 } elseif $(\exists P_j \in FCand_i, [root_j = MaxRoot_i \wedge rank_j = MinRank_i])$ {
8 $root_i := MaxRoot_i$;
9 $rank_i := MinRank_i + 1$;
10 $father_i := \max\{(D_j, P_j) \mid P_j \in FCand_i \wedge root_j = root_i \wedge rank_j = rank_i - 1\}$;
11 } elseif $(MinRank_i + 1 \leq n \wedge MaxChildren_i \geq 2 \wedge MaxRoot_i \neq (D_i, P_i))$ {
12 $root_i := MaxRoot_i$;
13 $rank_i := MinRank_i + 1$;
14 $father_i := \max\{(D_j, P_j) \mid P_j \in N_i \wedge root_j = root_i \wedge rank_j = rank_i - 1\}$;
15 } elseif $(\exists P_j \in FCand_i, [root_j = MaxRoot_i])$ {
16 $root_i := MaxRoot_i$;
17 $rank_i := \min\{rank_j \mid P_j \in FCand_i \wedge root_j = root_i\} + 1$;
18 $father_i := \max\{(D_j, P_j) \mid P_j \in FCand_i \wedge root_j = root_i \wedge rank_j = rank_i - 1\}$;
19 } elseif $(|FCand_i| \geq 1)$ {
20 $root_i := \max\{root_j \mid P_j \in FCand_i\}$;
21 $rank_i := \min\{rank_j \mid P_j \in FCand_i \wedge root_j = root_i\} + 1$;
22 $father_i := \max\{(D_j, P_j) \mid P_j \in FCand_i \wedge root_j = root_i \wedge rank_j = rank_i - 1\}$;
23 } else {
/* For singleton.*/
24 $root_i := \emptyset$; $rank_i := 0$; $father_i := (D_i, P_i)$;
25 }
}

Fig. 1. SSMLF: Self-stabilizing algorithm for construction of the maximal leafy forest

of P_i that can minimize their height in the tree by changing its father to P_i. The *expected number of children* is the number of its children and child-candidates. Each process P_i counts the number of expected number of children to make the tree leafy, and joins a tree as large as possible. That is, if P_i is the root and its tree is larger than the one of its neighbors, then all its neighbors will join the tree of P_i.

According to Definition 4, SSMLF constructs of a maximal leafy forest (MLF) of G by assigning its outputs following Definitions 5 and 6.

Definition 5. *Let $sdeg_i$ be the degree of process P_i in its tree, i.e., $sdeg_i \equiv |\{P_j \in N_i \mid father_j = (D_i, P_i)\}| + |\{P_j \in N_i \mid father_i = (D_j, P_j)\}|$.* □

Definition 6. *Let $T_k = (V_k, E_k)$ be a tree rooted at a process P_r where E_k is a set of links represented by the value of $father_i$ of each process P_i of $V_k \subseteq V$ and $sdeg_r \geq 3$ holds. Consider each process P_i such that $sdeg_i = 2$ on T_k. If $sdeg_f \geq 3$ and $sdeg_j \geq 3$ where $father_i = (D_f, P_f)$ and $father_j = (D_i, P_i)$, then T_k is leafy tree. If each tree $T_1, T_2, \cdots$ is disjoint and leafy on G, then the set $\{T_1, T_2, \cdots\}$ is a leafy forest F. If F is not a subgraph of any other leafy forest, then F is maximal leafy forest.* □

In order to evaluate its output variables, a process P_i uses several macros:

- $MaxRoot_i$ returns the largest value in the *root*-variables of P_i and its neighbors.
- $CCand_i$ returns the set of child-candidates of P_i.
- $CountMaxChildren_i$ returns the expected number of children of P_i.
- $FCand_i$ returns the *father candidates* of P_i, that is, the neighbors that P_i can choose in order to make its tree leafy, *i.e.*, process P_j such that $rank_j$ is not obviously inconsistent and that has a chance of holding $sdeg_j \geq 3$.
- $MinRank_i$ returns the *rank*-value of the (current or future) father of P_i in its tree. (If P_i is the root of its tree, then $MinRank_i$ returns -1 so that P_i sets $rank_i$ to 0.)

Now, we give more details about SSMLF. Consider a process P_i. In Lines 1-2, if the value of $root_i$ is obviously inconsistent, *i.e.*, $root_i > MAX$, then P_i resets its value to $\emptyset$. In Lines 3-4, P_i updates $MaxChildren_i$, if necessary. Then, if $root_i$ and $MaxChildren_i$ are correctly evaluated, P_i must choose its status among *root, internal node or leaf* of a tree and *singleton*, and updates its variables $root_i$, $rank_i$, and $father_i$ in consequence. That is what it does in Lines 5-24. Below, we detail these lines:

- In Lines 5-6, if $MaxChildren_i \geq 3$ and P_i can become a root of large tree, then P_i becomes a root.
- In Lines 7-10, P_i selects a process P_j such that the distance from the root to P_j is the minimum on a largest tree as $father_i$ only if P_j has a chance of holding $sdeg_j \geq 3$. That is, $P_j \in FCand_i$.
- In Lines 11-14, if P_i has a chance of holding $sdeg_i \geq 3$, then P_i selects a process P_j such that the distance from the root to P_j is the minimum on a largest tree as $father_i$ even if $sdeg_j < 3$, in order to make the tree leafy. By the condition of $MinRank_i + 1 \leq n$, P_i does not select a process P_j such that the value of $rank_j$ is obviously inconsistent.
- In Lines 15-18, if P_i does not have a chance of holding $sdeg_i \geq 3$, P_i selects a process P_j on a largest tree as $father_i$ only if P_j has a chance of holding $sdeg_j \geq 3$ by the condition $P_j \in FCand_i$.

– In Lines 19-22, if P_i cannot belong to a largest tree, P_i belongs to other tree.
– In Line 24, if P_i cannot belong to any tree, P_i becomes a singleton.

To show the correctness, we must first define the set of legitimate configurations of SSMLF. Such a definition is given in Definition 7 below:

Definition 7. *A configuration of* SSMLF *is legitimate if and only if each process P_i satisfies the following conditions:*

– *The connection by $father_i$ represents the maximal leafy forest.*
– *If P_i is on a tree T of the maximal leafy forest, then the value of $root_i$ of P_i represents the root of T.*
– *If P_i is not on any tree of the maximal leafy forest, then P_i is called a* singleton, *$father_i = (D_i, P_i)$ and $root_i = \emptyset$.*

By Λ_f, we denote a set of legitimate configuration. □

Theorem 2. *The algorithm* SSMLF *is self-stabilizing with respect to Λ_f.*

Proof Outline. The first part of the proof consists in proving that any terminal configuration of SSMLF is legitimate. So, in the following, we consider a configuration γ' where no process is privileged. In γ', each connected component T represented by the value of $father_i$ is a tree or a singleton on G. Additionally, there exists no process such that the value of $rank_i$ is obviously inconsistent in γ', *i.e.*, for each process P_i, $rank_i \leq n-1$ holds. Then, each tree is rooted at a process P_i such that $father_i = root_i = (D_i, P_i)$ and $sdeg_i = MaxChildren_i = D_i$, and P_0 such that $(D_0, P_0) = MAX$ is a root of a tree. Additionally, each singleton P_i holds $father_i = (D_i, P_i)$, $root_i = \emptyset$ and $sdeg_i = 0$. Other processes P_i selects a process P_j such that $sdeg_j \geq 3$ as its father if $sdeg_i < 3$, *i.e.*, each tree T represented by the values of *father*s is leafy, and the leafy forest is maximal.

The second part of the proof consists in proving that eventually the configuration becomes γ' by the algorithm SSMLF.

For any configuration γ and any computation starting from γ, each value of $root_i$ of each process P_i on G eventually become smaller than or equal to MAX, and P_0 such that $MAX = (D_0, P_0)$ decides the values of its each variable as a root of a tree and never change. After that, some processes P_i eventually form the tree $T_0 = (V_0, E_0)$ rooted at P_0 and never change the values of their each variable.

Let $G^1 = (V^1, E^1)$ be an induced subgraph by $V \setminus V_0$. Let (D_1^-, P_1^-) be the maximum couple among the processes on G^1 which are not neighboring to any process in V_0. If $D_1^- \geq 3$, P_1^- eventually fixes the values of its variables to be a root forever. After that, some processes P_i eventually form the tree $T_1 = (V_1, E_1)$ rooted at P_1^- and stop changing the values of their variables.

By repeating this discussion, we have a series of trees $T_0 = (V_0, E_0)$, $T_1 = (V_1, E_1)$, $\cdots$, $T_k = (V_k, E_k)$, where each process in $V_0, V_1, \cdots, V_k$ fixes the values of all its variables. Let G^k be an induced subgraph by $V \setminus \{V_0 \cup V_1 \cup .. \cup V_k\}$. If $D_k^- < 3$ holds where the maximum couple on G^k among the processes which are not neighboring to any process in $\{V_0 \cup V_1 \cup \cdots \cup V_k\}$, processes on G^k cannot form any leafy tree, and they become singletons. Then, no process is privileged. □

Theorem 3. *The time complexity of algorithm* SSMLF *is $O(n^2)$ rounds.*

3.2 The Third Layer: Modification of Edge Cost

The third layer algorithm SSTN computes a network cost from the MLF computed by the second layer. In the fourth layer, the minimum spanning tree is computed based on this cost. The minimum spanning tree is the approximate solution of the MLST problem. Then, the edges in any tree must include the minimum spanning tree, and the *intra-tree* edges such that both endpoints are in the same tree must not be in the minimum spanning tree. Additionally, we would like to make the number of the connector edges between each tree as small as possible. The connector edges are selected from *inter-tree* edges such that its endpoints are not in the same tree.

Formal description of this layer SSMLF is shown in Fig. 2.

Definition 8. *A configuration of* SSTN *is legitimate if and only if each edge satisfies the following three conditions.*

- *If e is a tree edge, then* $W(P_i)[P_j]$ *is* 0,
- *If e is an intra-tree edge, then* $W(P_i)[P_j]$ *is* ∞, *and*
- *If e is an inter-tree edge, then* $W(P_i)[P_j]$ *is* 1.

By Λ_t, *we denote a set of legitimate configuration.* □

Constant (Input)
N_i: the set of neighbors on G.
$root_i$: ID of the root of tree T to which P_i belongs on the MLF (output from Layer 2).
$father_i$: ID of the father of P_i in T (output from Layer 2).
Variable (Output)
$W(P_i)[P_j]$: new cost of the edge between P_i and $P_j \in N_i$.
Algorithm for process P_i**:**
do *forever*{
1 if $(\exists P_j \in N_i[root_i \neq root_j \wedge W(P_i)[P_j] \neq 1])$ {
2 $W(P_i)[P_j] := 1$; /* For Inter-tree edge */
3 } elseif $(\exists P_j \in N_i[root_i = root_j \wedge (father_i \neq P_j \wedge father_j \neq P_i) \wedge W(P_i)[P_j] \neq \infty])$ {
4 $W(P_i)[P_j] := \infty$; /* For Intra-tree edge */
5 } elseif $(\exists P_j \in N_i[root_i = root_j \wedge (father_i = P_j \vee father_j = P_i) \wedge W(P_i)[P_j] \neq 0])$ {
6 $W(P_i)[P_j] := 0$; /* For Tree edge */
7 }
}

Fig. 2. SSTN: Self-stabilizing algorithm for transforming the network

Theorem 4. *The algorithm* SSTN *is self-stabilizing with respect to* Λ_t. *The time complexity of algorithm* SSTN *is* $O(n)$ *rounds.*

Each of the four layers stabilize in at most $O(n^2)$ rounds, hence we can conclude:

Theorem 5. *The algorithm* SSMLST *is a self-stabilizing approximation algorithm for the MLST problem with approximation ratio 3. Its time complexity is* $O(n^2)$ *rounds.*

4 Conclusion

In this paper, we proposed a self-stabilizing distributed approximation algorithm for the MLST problem in arbitrary topology networks with approximation ratio 3. However, there exists a sequential solution [1] proposed by Solis-Oba that has approximation ratio 2. Investigating the trade-off between approximation ratio and complexity of the self-stabilizing mechanism to achieve it is an immediate future work. Also, we would like to mention the importance to complement the self-stabilizing abilities of a distributed algorithm with some additional *safety* properties that are guaranteed when the permanent and intermittent failures that hit the system satisfy some conditions.

References

1. Solis-Oba, R.: 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In: Proceedings of the 6th Annual European Symposium on Algorithms. LNCS 1461 (1998) 441–452
2. Tixeuil, S.: Self-stabilizing Algorithms. Chapman & Hall/CRC Applied Algorithms and Data Structures. In: Algorithms and Theory of Computation Handbook, Second Edition. CRC Press, Taylor & Francis Group (November 2009) 26.1–26.45
3. Galbiati, G., Maffioli, F., Morzenti, A.: A short note on the approximability of the maximum leaves spanning tree problem. Information Processing Letters **52**(1) (1994) 45–49
4. Lu, H.I., Ravi, R.: Approximating maximum leaf spanning trees in almost linear time. Journal of algorithms **29** (1998) 132–141
5. Blin, L., Potop-Butucaru, M., Rovedakis, S.: Self-stabilizing minimum-degree spanning tree within one from the optimal degree. In: Proceedings of the 23th IEEE International Parallel and Distributed Processing Symposium. (2009)
6. Blin, L., Potop-Butucaru, M.G., Rovedakis, S., Tixeuil, S.: A new self-stabilizing minimum spanning tree construction with loop-free property. In: Proceedings of the 23rd International Symposium on Distributed Computing. LNCS 5805 (2009) 407–422
7. Butelle, F., Lavault, C., Bui, M.: A uniform self-stabilizing minimum diameter tree algorithm. In: Proceedings of the 9th International Workshop on Distributed Algorithms. LNCS 972 (1995) 257–272
8. Guha, S., Khuller, S.: Approximation Algorithms for Connected Dominating Sets. Algorithmica **20** (1998) 347–387
9. Kamei, S., Kakugawa, H.: A self-stabilizing distributed approximation algorithm for the minimum connected dominating set. In: Proceedings of the 9th IPDPS Workshop on Advances in Parallel and Distributed Computational Models. (2007) 224
10. Raei, H., Tabibzadeh, M., Ahmadipoor, B., Saei, S.: A self-stabilizing distributed algorithm for minimum connected dominating sets in wireless sensor networks with different transmission ranges. In: Proceedings of the 11th International Conference on Advanced Communication Technology. (2009) 526–530
11. Dolev, S.: Self-Stabilization. The MIT Press (2000)
12. Datta, A.K., Larmore, L.L., Vemula, P.: Self-stabilizing leader election in optimal space. In: Proceedings of the 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems. LNCS 5340 (2008) 109 – 123