J Comb Optim (2013) 25:430–459
DOI 10.1007/s10878-011-9383-5

A self-stabilizing 3-approximation for the maximum leaf spanning tree problem in arbitrary networks

Sayaka Kamei · Hirotsugu Kakugawa · Stéphane Devismes · Sébastien Tixeuil

Published online: 27 January 2011

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 in arbitrary topology networks. Our algorithm is the first self-stabilizing protocol that is specifically designed to approximate an MLST. It builds a solution whose number of leaves is at least $1/3$ of the maximum possible in arbitrary graphs. The time complexity of our algorithm is $O(n^2)$ rounds.

Keywords Self-stabilization · Approximation · Maximum leaf spanning tree · Fault-tolerance

S. Kamei (✉)
Dept. of Information Engineering, Graduate School of Engineering, Hiroshima University, Hiroshima, Japan
e-mail: s-kamei@se.hiroshima-u.ac.jp

H. Kakugawa
Graduate School of Information Science and Technology, Osaka University, Osaka, Japan
e-mail: kakugawa@ist.osaka-u.ac.jp

S. Devismes
Université Joseph Fourier, Grenoble, France
e-mail: Stephane.Devismes@imag.fr

S. Tixeuil
UPMC Sorbonne Universités, Paris, France
e-mail: Sebastien.Tixeuil@lip6.fr

S. Tixeuil
Institut Universitaire de France, Paris, France

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 permitting them to autonomously construct routing and communication primitives that are used by higher level applications. The construction of virtual backbone 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 $G = (V, E)$ 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 (Solis-Oba 1998). Unfortunately, finding the MLST is known to be NP-complete (Garey 1979).

One of the most versatile techniques to ensure forward recovery of distributed systems and networks is that of *self-stabilization* (Dijkstra 1974; Dolev 2000; Tixeuil 2009). 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.). By such property, the self-stabilizing systems do not need any initialization of states of each process and each link.

1.1 Related works

Galbiati et al. (1994) proved that the MLST problem is MAX-SNP-hard, *i.e.*, there exists $\epsilon \in]0 \dots 1[$ such that finding approximation algorithm[1] with approximation ratio $1 + \epsilon$ is NP-hard. Solis-Oba (1998) proposed a 2-approximation algorithm, and Lu and Ravi (1998) proposed a 3-approximation algorithm. Note that none of these algorithms is distributed, not to mention self-stabilizing.

Spanning tree construction is one of the main problems studied in self-stabilizing literature (Gartner 2003). One of the main trends recently in this topic is to provide self-stabilizing protocols for constrained variants of the spanning tree problem, *e.g.* the minimum degree spanning tree (Blin et al. 2009), the minimum weight spanning tree (Blin et al. 2009), the minimum diameter spanning tree (Butelle et al. 1995), etc. Nevertheless, none of these metrics gives guarantees on the number of leaves.

There exist self-stabilizing approximation algorithms for finding the minimum CDS. Kamei and Kakugawa (2007) proposed a self-stabilizing 7.6-approximation

[1] 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.

algorithm for the CDS problem in unit disk graphs. However, this algorithm does not guarantee any approximation ratio for arbitrary networks. The subsequent work of Raei et al. (2009) 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. Again, this algorithm does not guarantee any approximation ratio for arbitrary networks. They guarantee the approximation ratio of the minimum CDS, that is, they guarantee the number of the member of the CDS. However, they do not guarantee the approximation ratio of the MLST problem. That is, they cannot guarantee the number of leaves in the CDS.

1.2 Contribution

We propose a fully distributed self-stabilizing approximation algorithm for the MLST problem in arbitrary networks. Its time complexity is $O(n^2)$ rounds. To our knowledge, our algorithm is the first self-stabilizing protocol that is especially designed to approximate a MLST. Namely, it constructs a spanning tree whose number of leaves is greater or equal to $1/3$ of the maximum possible.

Our algorithm can be used to construct a virtual backbone in multihop wireless ad hoc or sensor networks. Its effective approximation ratio permits to significantly improve the load and the energy consumed by the virtual backbone.

Moreover, our scheme being designed for arbitrary topologies, it is useful even in networks that cannot be modeled by (generalized) disk graphs such as wired networks.

1.3 Roadmap

This paper is organized as follows: In Sect. 2, we formally describe the system model and the distributed MLST problem. In Sect. 3, we present our self-stabilizing approximation algorithm for the distributed MLST problem, we prove its correctness, and analyze its time complexity. Concluding remarks can be found in Sect. 4.

2 Preliminaries

Let $V = \{P_1, P_2, \ldots, P_n\}$ be a set of n processes and $E \subseteq V \times V$ be a set of bidirectional communication links in a distributed system. Each link is an unordered pair of distinct processes. The system topology is represented by the 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.

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 constant. We assume that the maximum degree of G is at least 3. If the maximum degree of G is less than 3, then it is clear that G is the MLST of G. We define the *distance* between P_i and P_j as the number of the edges of the shortest path between them.

The local state of a process is defined by the value of all its local variables. A *configuration* of the system is an instance of the local states of all processes. The set of all configurations is denoted by Γ.

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

We say that P_i is *privileged* in a configuration γ if and only if P_i can change its local state by the algorithm 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 its local state based on 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 to execute an atomic step in parallel. Weakly fair means that any continuously privileged process is eventually selected by the daemon.

For any configuration γ, if γ' can be obtained from γ (according to the algorithm and the daemon), then we denote this transition 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) 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, the system eventually reaches a configuration of 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 an 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 A *maximum leaf spanning tree* is a spanning tree whose number of leaves is maximum.

We consider the MLST problem in distributed systems, so we assume that each process does not know the global information on 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. The *distributed maximum leaf spanning tree problem* is defined as follows:

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

3 The algorithm

Our algorithm SSMLST is based on a sequential approximation algorithm proposed in (Lu and Ravi 1998).

We call *tree* any connected acyclic subgraph T of G containing more than one process. We construct disjoint trees $T_1, T_2, \ldots$, where $T_i = (V_i, E_i)$, $V = V_1 \cup V_2 \cup \ldots$, $|V_i| > 1$, and $V_i \cap V_j = \emptyset$ for any i and j. We call *forest* any set of trees $\{T_1, T_2, \ldots\}$. 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 having degree k in G, and $\bar{d}_k(G)$ be the set of nodes having degree at least k in G.

Definition 4 (Lu and Ravi 1998) 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 $\bar{d}_3(T)$, then T is said to be *leafy*. Let $T_1, T_2, \ldots$ be disjoint trees on G. If each $T_1, T_2, \ldots$ is leafy, then $F = \{T_1, T_2, \ldots\}$ is a *leafy forest*. If F is not a subgraph of any other leafy forest of G, then F is called a *maximal leafy forest*.

Lu and Ravi (1998) showed the following theorem.

Theorem 1 (Lu and Ravi 1998) *Let F be a maximal leafy forest of G, and 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, 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 by Theorem 1.

Figure 1 shows an example of a legitimate configuration obtained using SSMLST. Each circle represents a process, and the number in the circle is the process ID. The arrows represent the MLF, and white circles represent singletons. Each number out of a circle gives the distance from the process to the root of its tree. The dashed edges are edges of T_{ml} connecting trees of the MLF and singletons. Finally, each circle with double effect is leaf of the MLST.

The proposed algorithm is a fair composition (Dolev 2000) of four layers:

1. In the first layer, each process P_i computes its degree D_i in G and the maximum pair of degree and ID in G noted $MAX = (D_k, P_k)$. This pair is evaluated according to the following order relation: for each process P_i and P_j, $(D_i, P_i) > (D_j, P_j) \equiv [D_i > D_j \vee (D_i = D_j \wedge P_i > P_j)]$. For this layer, we can use any self-stabilizing leader election algorithm for arbitrary networks, for example (Datta et al. 2008).[2] In such an algorithm, the process with the minimum or

[2] The time complexity of this algorithm is $O(n)$ rounds.

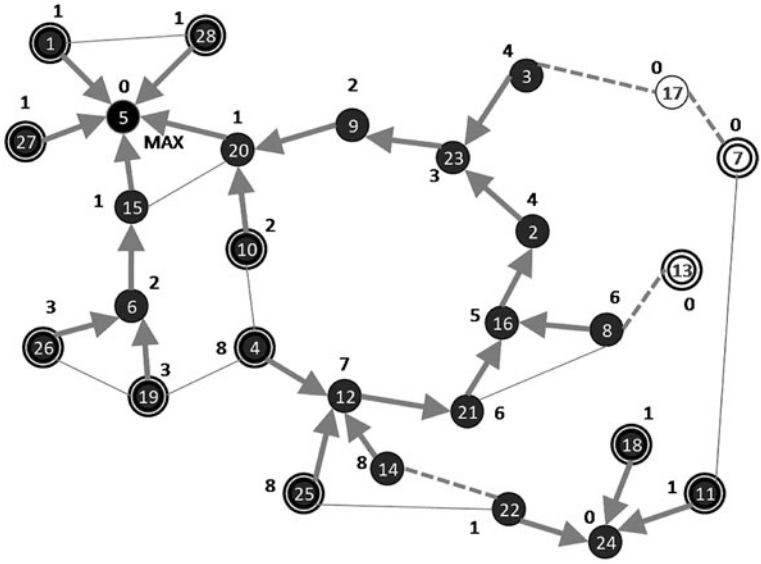

Fig. 1 Example of the MLST

the maximum ID is elected as a leader. It can be modified to elect the process with the maximum pair of degree and ID for our purpose.
2. The second layer SSMLF (presented in Sect. 3.1) computes an MLF of G.
3. The third layer SSTN (presented in Sect. 3.2) assigns 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 force T_{ml} to include the MLF. For this layer, we can use one of existing self-stabilizing algorithms, *e.g.*, (Blin et al. 2009).[3]

3.1 The second layer: construction of the maximal leafy forest

We now propose a self-stabilizing algorithm called SSMLF that constructs a maximal leafy forest (MLF) of G. The formal description of SSMLF is shown in Fig. 2.

In SSMLF, we assume that the following inputs are available at each process P_i:

– N_i, the set of P_i's neighbors in G,
– D_i, the degree of P_i in G (an output from the first layer), and
– MAX, the maximum pair of degree and ID in G (an output from the first layer).

Based on these inputs, each process P_i computes the value of the following output variables:

– $root_i$ is set to $\emptyset$ if P_i is a singleton. Otherwise, P_i belongs to some tree T and $root_i$ is set to (D_r, P_r), where P_r is the root of the tree of P_i.

[3]The time complexity of this algorithm is $O(n^2)$ rounds.

Constants (Inputs)
N_i: the set of neighbors of P_i in G.
D_i: the degree of P_i in G (an output from the first layer).
MAX: the maximum pair of degree and ID in G (an output from the first layer).
Variables (Outputs)
$root_i = (D_r, P_r)(\emptyset \leq root_i \leq MAX)$: P_r is the root of tree T to which P_i belongs.
$father_i$: the pair (D_j, P_j) of the father P_j of P_i in T.
$rank_i$: the distance from the root to P_i in T.
$MaxChildren_i$: the number of children and child-candidates in T.
Macros
$MaxRoot_i = \max\{root_j,\ (D_i, P_i) \mid P_j \in N_i\}$
$MinRank_i = \begin{cases} -1 & \text{(In case that } MaxRoot_i = (D_i, P_i)\text{)} \\ \min\{rank_j \mid P_j \in N_i \wedge root_j = MaxRoot_i\} & \text{(otherwise)} \end{cases}$
$CCand_i =$
$\{P_j \in N_i \mid root_j = \emptyset \vee root_j < MaxRoot_i \vee (root_j = MaxRoot_i \wedge rank_j > MinRank_i + 2)\}$
$CountMaxChildren_i = \begin{cases} D_i & \text{(In case that } (D_i, P_i) = MAX\text{)} \\ |\{P_j \in N_i \mid father_j = (D_i, P_i)\}| + |CCand_i| & \text{(otherwise)} \end{cases}$
$FCand_i = \{P_j \in N_i \mid rank_j + 1 \leq n \wedge (MaxChildren_j \geq 3 \vee$
$(MaxChildren_j = 2 \wedge father_j \neq (D_j, P_j) \wedge root_j > (D_j, P_j)))\}$
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 singletons.*/
24 $root_i := \emptyset$; $rank_i := 0$; $father_i := (D_i, P_i)$;
25 }
}

Fig. 2 SSMLF: Self-Stabilizing algorithm for construction of the Maximal Leafy Forest

– $father_i$ is set to (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 other 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 outgoing link to $father_i$ as its parent-link in its tree.
- $rank_i$ is set to the distance from P_i to the root of its tree.
- $MaxChildren_i$ is set to the *expected number of children* of P_i in its tree. (We shall explain that later.)

We now explain how we compute these outputs. In the explanations, we call *large tree* any tree rooted at a process with a large pair of degree and ID in G. That is, if there exists two trees rooted P_i and P_j respectively and $(D_i, P_i) > (D_j, P_j)$, then the tree rooted at P_i is larger than at P_j. Also, we call a *child-candidate* of process P_i any neighbor of P_i that may become a child of P_i in the future, *e.g.*, singleton, any process belonging to a tree that is not larger than the tree of P_i, or any process belonging to the tree of P_i that can minimize its rank 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 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 its tree.

According to Definition 4, SSMLF constructs of a maximal leafy forest (MLF) of G. Then, SSMLF assigns output values of each process 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 some process P_r such that $sdeg_r \geq 3$ and E_k is a set of links represented by the value of $father_i$ of each process P_i of $V_k \subseteq V$.

- Consider each process P_i such that $sdeg_i = 2$ in 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 $T_1, T_2, \ldots$ is disjoint and leafy in G, then the set $\{T_1, T_2, \ldots\}$ is a leafy forest F.
- If F is not a subgraph of any other leafy forest in G, 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 among 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 as father in order to make its tree leafy, *i.e.*, process P_j such that $rank_j$ is not obviously inconsistent ($rank_j \leq n - 1$) 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. (If P_i is the root of its tree, then $MinRank_i$ returns -1 so that P_i sets $rank_i$ to 0.)

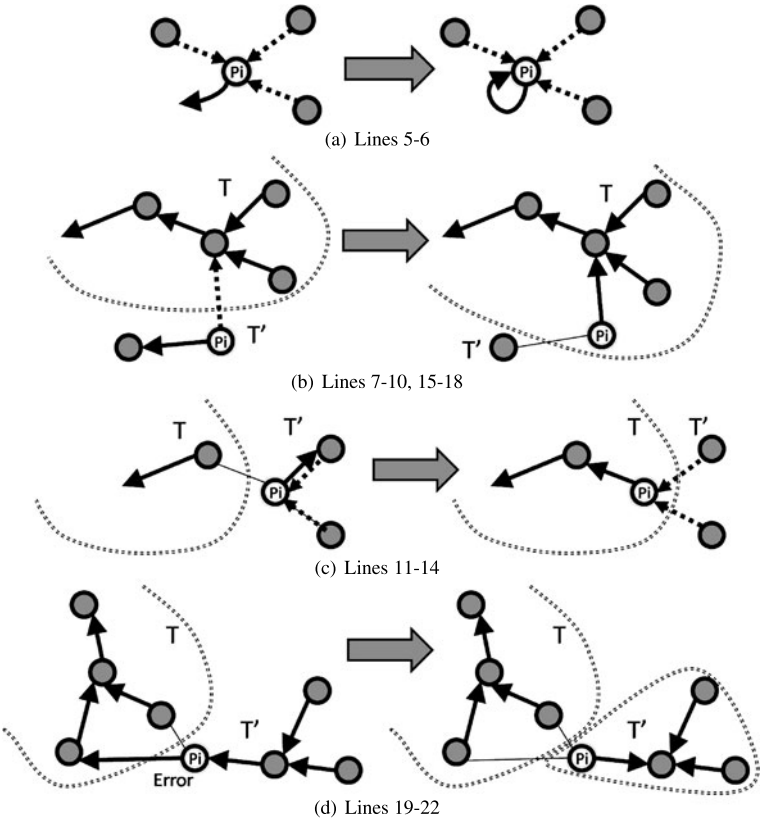

Fig. 3 Outline of the proposed algorithm

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. In Lines 5–24, if $root_i$ and $MaxChildren_i$ are correctly evaluated, P_i chooses 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. Then, each process joins a neighboring tree as large as possible, and make the height of its tree as small as possible.

We now give more detail about Lines 5–24. Our explanations are based on Fig. 3. In this figure, bold arrows represent father pointers, dashed arrows represent child-candidates, and dashed curve lines represent border of trees. Moreover, we assume that the tree T is a larger tree than T'. Below, we detail Lines 5–24:

- 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. (See Fig. 3(a).)
- In Lines 7–10, P_i selects as $father_i$ a neighbor P_j in a largest tree whose distance to the root is minimum only if P_j has a chance of holding $sdeg_j \geq 3$. That is, $P_j \in FCand_i$. (See Fig. 3(b).)

– In Lines 11–14, if P_i has a chance of holding $sdeg_i \geq 3$, then P_i selects as $father_i$ a neighbor P_j in a largest tree whose distance to the root is minimum (even if $sdeg_j < 3$) in order to make the tree leafy. Note that, by the condition of $MinRank_i + 1 \leq n$, P_i does not select a process P_j whose value of $rank_j$ is obviously inconsistent. (See Fig. 3(c).)
– In Lines 15–18, if P_i does not have a chance of holding $sdeg_i \geq 3$, P_i selects as $father_i$ a neighbor P_j in a largest tree only if P_j has a chance of holding $sdeg_j \geq 3$ by the condition $P_j \in FCand_i$. (See Fig. 3(b).)
– In Lines 19–22, if P_i cannot belong to a largest tree, P_i belongs to another tree. (See Fig. 3(d).)
– In Line 24, if P_i cannot belong to any tree, P_i becomes a singleton.

We now show the correctness of our algorithm. The proof of correctness is based on the definition of the 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 in a tree T of the maximal leafy forest, then the value of $root_i$ is the pair of degree and ID of the root of T.
– If P_i is not in any tree of the maximal leafy forest, then P_i is a *singleton*, that is $father_i = (D_i, P_i)$ and $root_i = \emptyset$.

By Λ_f, we denote a set of legitimate configuration.

We now prove the correctness of SSMLF. 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 the following lemmas, we use the notions of *singleton* and *root* defined below:

Definition 8 Let a *singleton* be any process P_i such that $father_i = (D_i, P_i) \wedge root_i = \emptyset$. Let a *root of a tree* be any process P_i such that $root_i = father_i = (D_i, P_i)$.

To prove that any terminal configuration is legitimate, first, we show that, in γ', each connected component represented by the value of $father_i$ is a tree or a singleton in G (Lemma 1). Next, we show that there exists no process such that the value of $rank_i$ is obviously inconsistent in γ' (Lemma 2). 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 (Lemmas 3 and 5). Additionally, each singleton P_i has $father_i = (D_i, P_i)$, $root_i = \emptyset$, and $sdeg_i = 0$ (Lemma 4). Other processes P_i selects a process P_j such that $sdeg_j \geq 3$ as its father if $sdeg_i < 3$ (Lemmas 6, 7 and 8). That is, each tree T represented by the values of $father$s is leafy. Finally, we show that the leafy forest is maximal (Lemma 9). Now, let start proving each lemma.

According to the algorithm, in γ', if some process P_i is a root or a singleton, then $rank_i = 0$. In other cases, $rank_i = rank_j + 1$ and $father_i = (D_j, P_j)$, which means that the rank of P_i is the rank of its father plus one. This property prevents the existence of father-link cycle in γ', as proven in the following lemma:

Lemma 1 *Each connected component T represented by the value of $father_i$ is a tree or a singleton in G in γ'.*

Proof By definition of the algorithm, each value of $father_i$ is a self-loop or a neighbor. So, to show that processes construct trees or singletons, we prove that there is no loop of length at least 2 in γ'.

Suppose that there exists a loop $P_0, \ldots, P_k$ of length at least two in γ'. Then, for each process P_i with $0 \leq i \leq k$, $father_i = (D_{(i+1)mod(k+1)}, P_{(i+1)mod(k+1)})$ holds. By Lines 10, 14, 18 and 22, each process selects the neighbor $father_i$ which satisfies $rank_{father_i} = rank_{(i+1)mod(k+1)} = rank_i - 1$ in γ'. Therefore, if there exists such a loop, at least one process in the loop is privileged by one of these lines, a contradiction. Therefore, in γ', there exists no loop of length at least 2.

Therefore, each connected component is a singleton or a tree in G in γ'. □

According to the algorithm, if $rank_i > n-1$, then P_i is privileged. Hence follows:

Lemma 2 *In γ', for each process P_i, $rank_i \leq n-1$ holds.*

Proof By Lemma 1, each connected component by the value of $father_i$ is a tree or a singleton.

Consider first any tree T in γ'. Let P_i be the root of T. The value of $rank_i$ is 0 due to Line 6. Additionally, by Lines 10, 14, 18 and 22, the value of $rank_j$ of each other process P_j of T is the number of edges in the path from P_i to P_j. Therefore, we have $rank_j \leq n-1$.

Consider now a singleton P_i in γ'. The value of $rank_i$ is 0 due to Line 24. □

The following lemma can be deduced from the definition of $FCand_i$.

Lemma 3 *Let P_0 be the process such that $MAX = (D_0, P_0)$. Let $P_i \in N_0$. In γ', $father_i = (D_0, P_0)$ and $sdeg_0 = D_0$ hold.*

Proof As we assume that the maximum degree is at least 3 in G, $D_0 \geq 3$ holds. By Lines 3 and 4, $MaxChildren_0 = D_0 \geq 3$ holds because $(D_0, P_0) = MAX$. By definition of $MaxRoot_0$, $MaxRoot_0 = (D_0, P_0)$ holds because $root_i \leq MAX = (D_0, P_0)$ for any process $P_i \in N_0$ by Lines 1 and 2. Thus, the condition of Line 5 is true at P_0. Therefore, by Line 6, $root_0 = (D_0, P_0)$, $rank_0 = 0$ and $father_0 = (D_0, P_0)$ hold. Moreover, $P_0 \in FCand_i$ holds for any neighbor $P_i \in N_0$ because $rank_0 + 1 = 0 + 1 \leq n \wedge MaxChildren_0 \geq 3$.

Suppose that there exists a neighbor $P_i \in N_0$ such that $father_i \neq (D_0, P_0)$ in γ'. By definition of $MaxRoot_i$, $MaxRoot_i = root_0 = (D_0, P_0) = MAX$. As $MaxRoot_i = MAX > (D_i, P_i)$, the condition of Line 5 is false at P_i. For the same reason and by definition of $MinRank_i$, $MinRank_i = 0$. As $P_0 \in FCand_i$, $MaxRoot_i = root_0$, and $MinRank_i = rank_0 = 0$, the condition of Line 7 is true at P_i. So, P_i is privileged in Line 10 because $father_i \neq (D_0, P_0) = MaxRoot_i$, a contradiction.

Therefore, each neighbor P_i of P_0 in G satisfies $father_i = (D_0, P_0)$. Finally, as the number of neighbors of P_0 in G is D_0, $sdeg_0 = D_0$ by Definition 5. □

In the algorithm, a process becomes a singleton only if it has no chance to belong to any leafy tree, thus:

Lemma 4 *In γ', for each process P_i such that $root_i = \emptyset$ holds (i.e., a singleton), $MaxChildren_i < 3 \wedge sdeg_i = 0$ holds.*

Proof Suppose by contradiction that $MaxChildren_i \geq 3$ or $sdeg_i \neq 0$ hold in γ'.

First, suppose that $father_i = (D_i, P_i)$ holds at P_i. The value of $father_i$ becomes (D_i, P_i) only at Lines 6 and 24. As $root_i = \emptyset$, Line 24 is the only possibility, *i.e.*, the conditions of Lines 5, 7, 11, 15 and 19 are false at P_i. By Lemma 2, $rank_j \leq n - 1$ holds at each neighbor $P_j \in N_i$. Thus, $MinRank_i + 1 \leq n$ holds by the definition of $MinRank_i$.

- Case 1: Assume that $MaxChildren_i \geq 3$. As the condition of Line 5 is false and $MaxChildren_i \geq 3$ holds, $MaxRoot_i \neq (D_i, P_i)$ holds at P_i. As $MinRank_i + 1 \leq n \wedge MaxChildren_i \geq 3 \wedge MaxRoot_i \neq (D_i, P_i)$, the condition of Line 11 is true at P_i. This is a contradiction because the condition of Line 11 is false at P_i by assumption. Thus, this case does not occur and we have $MaxChildren_i < 3$.
- Case 2: Assume that $sdeg_i \neq 0$.
  - ◇ Suppose that $sdeg_i \geq 3$ holds. As $father_i = (D_i, P_i)$, $MaxChildren_i \geq 3$ by Definition 5. This is a contradiction because $MaxChildren_i < 3$ by Case 1.
  - ◇ Suppose that $0 < sdeg_i \leq 2$ holds. As $father_i = (D_i, P_i)$, there exists at least one process $P_j \in N_i$ such that $father_j = (D_i, P_i)$ by Definition 5. The value of $father_j$ takes (D_i, P_i) by Lines 10, 14, 18 and 22, but not Lines 6 and 24. By these lines, $root_j = \emptyset$ holds because $root_i = \emptyset$. As $father_j = (D_i, P_i)$, $P_i \in FCand_j$ holds for execution of Lines 10, 18 and 22, or the condition of Line 11 is true at P_j for execution of Line 14.
    - Suppose that $P_i \in FCand_j$ holds. As $father_i = (D_i, P_i)$ holds, $MaxChildren_i \geq 3$ holds at P_i by definition of $FCand_j$. However, by Case 1, $MaxChildren_i < 3$ holds, a contradiction.
    - Suppose that the condition of Line 11 is true at P_j. Then, $MaxChildren_j \geq 2 \wedge MaxRoot_j \neq (D_j, P_j)$ holds. Thus, by definition of $MaxRoot_j$, $MaxRoot_j > (D_j, P_j)$. By Line 12, $root_j = MaxRoot_j$. As $root_j = \emptyset$ holds, $root_j = \emptyset = MaxRoot_j > (D_j, P_j)$, a contradiction.

Therefore, for each P_i such that $father_i = (D_i, P_i)$ and $root_i = \emptyset$, $MaxChildren_i < 3 \wedge sdeg_i = 0$ holds.

Next, suppose that $father_i \neq (D_i, P_i)$ holds at P_i, *i.e.*, $father_i = (D_j, P_j)$ for some $P_j \in N_i$. This case occurs at Lines 10, 14, 18 and 22 but not Lines 6 and 24. Then, by Lines 10, 14, 18 and 22, $root_i = root_j$ holds. Therefore, we have $root_i = root_j = \emptyset$. Consider a maximal path $P_i, P_j, P_k, \ldots, P_l, P_m$ where $father_i = (D_j, P_j)$, $father_j = (D_k, P_k), \ldots, father_l = (D_m, P_m)$. By Lemma 1, there exists no loop by the values of *father*s whose length is greater or equal to 2. Therefore, $father_m = (D_m, P_m)$ and $root_m = \emptyset$ holds at P_m. However, from the above discussion for the case of $father_m = (D_m, P_m)$ and $root_m = \emptyset$, $sdeg_m = 0$ holds. By Definition 5, because $sdeg_m = 0$ and $father_m = (D_m, P_m)$, there exists no process $P_l \in N_m$ such that $father_l = (D_m, P_m)$. Therefore, there exists no such a path, and there exists no process P_i such that $father_i \neq (D_i, P_i)$ and $root_i = \emptyset$ in γ', a contradiction. □

Assume that there exists a process P_i such that $father_i = (D_i, P_i) \neq MAX \wedge root_i \neq \emptyset \wedge MaxChildren_i \neq sdeg_i$. Then, there is a process in $MaxChildren_i$ that is privileged to become a child of P_i. As a consequence, we have:

Lemma 5 *In γ', for each process P_i such that $father_i = (D_i, P_i) \neq MAX \wedge root_i \neq \emptyset$ holds, $MaxChildren_i = sdeg_i$ holds.*

Proof First, we observe some properties on the local variables of P_i. The value of $father_i$ becomes (D_i, P_i) only at Lines 6 and 24. As $root_i \neq \emptyset$, Line 6 is the only possibility so that this case occurs. Thus, the condition of Line 5 must be true so that $father_i = (D_i, P_i) \wedge root_i \neq \emptyset$ holds (otherwise, the value of the two variables take other values). Thus, $MaxChildren_i \geq 3$ and $MaxRoot_i = (D_i, P_i)$ hold. Then:

– $MinRank_i = -1$

holds by definition of $MinRank_i$. By definition of $MaxRoot_j$ for every $P_j \in N_i$ and Line 6, we have:

– $MaxRoot_j \geq root_i = (D_i, P_i) = MaxRoot_i$ and
– $rank_i = 0$.

As $rank_i + 1 = 0 + 1 \leq n$ and $MaxChildren_i \geq 3$, by definition of $FCand_j$, we have:

– $P_i \in FCand_j$ at every $P_j \in N_i$, *i.e.*,
– $|FCand_j| \geq 1$.

Assume that the condition of Line 5 is true at every $P_j \in N_i$. Then, $MaxChildren_j \geq 3 \wedge MaxRoot_j = (D_j, P_j) = root_j$ by Line 6 in γ'. However, by the definitions of $MaxRoot_j$ and $MaxRoot_i$, $MaxRoot_j \geq root_i = (D_i, P_i) = MaxRoot_i$ and $MaxRoot_i \geq root_j = (D_j, P_j) = MaxRoot_j$ holds, *i.e.*, $(D_i, P_i) = (D_j, P_j)$, a contradiction. Therefore, the condition of Line 5 is false at every $P_j \in N_i$.

Based on these observation, we show the lemma by contradiction: Assume that $MaxChildren_i \neq sdeg_i$ for some process P_i such that $root_i = father_i = (D_i, P_i)$ in γ'. By Definition 5, the value of $sdeg_i$ is the number of children if $father_i = (D_i, P_i)$. By Line 4, the value of $MaxChildren_i$ is the number of children of P_i plus the size of the set $CCand_i$. Therefore, by assumption $MaxChildren_i \neq sdeg_i$, $|CCand_i| > 0$ holds, which implies that there exists $P_j \in CCand_i$ such that $P_j \in N_i$ and $root_j = \emptyset \vee root_j < (D_i, P_i) \vee (root_j = (D_i, P_i) \wedge rank_j > 1)$ because $MaxRoot_i = (D_i, P_i)$ and $MinRank_i = -1$ hold. Below, we check this condition.

– Case 1: Assume that $root_j = \emptyset$.
  ◇ Suppose that one of the conditions of Lines 7, 11 or 15 is true at P_j. Then, P_j is privileged in Lines 8, 12 or 16 because $MaxRoot_j \neq root_j = \emptyset$, a contradiction.
  ◇ Suppose otherwise, then the condition of Line 19 is true because $|FCand_j| \geq 1$. Then, $\max\{root_k \mid P_k \in FCand_j\} \neq \emptyset$ holds because $root_i \neq \emptyset$ and $P_i \in FCand_j$. Therefore, P_j is privileged in Line 20 because $root_j = \emptyset$, a contradiction.

  Thus, this case does not occur, and we have $root_j \neq \emptyset$.
– Case 2: Assume that $root_j < (D_i, P_i)$. As $MaxRoot_j \geq (D_i, P_i) = root_i$ holds, we have $root_j < (D_i, P_i) = root_i \leq MaxRoot_j$, which implies that $root_j \neq MaxRoot_j$ and $root_j < root_i$.

- ◇ Assume that one of the conditions of Lines 7, 11 or 15 is true at P_j. Then, P_j is privileged in Lines 8, 12 or 16, a contradiction.
- ◇ Assume otherwise, *i.e.*, the conditions of Lines 7, 11 and 15 are false at P_j. Then, as $P_i \in FCand_j$ and $root_j < root_i$, $root_j < root_i \leq \max\{root_l \mid P_l \in FCand_j\}$ holds. Thus, we have $root_j \neq \max\{root_l \mid P_l \in FCand_j\}$ and P_j is privileged in Line 20, a contradiction.

– Case 3: Assume that $root_j = (D_i, P_i) \wedge rank_j > 1$. By assumption $root_i = (D_i, P_i) \leq MaxRoot_j$ and $root_j = (D_i, P_i)$, we have $root_j = root_i = (D_i, P_i) \leq MaxRoot_j$.
  - ◇ Assume that $MaxRoot_j = root_j$, *i.e.*, $root_i = root_j = MaxRoot_j$ holds. Then, $MinRank_j \equiv \min\{rank_k \mid P_k \in N_j \wedge root_k = MaxRoot_j\} = rank_i = 0$. Thus, $rank_j > 1 = MinRank_j + 1$ holds, *i.e.*, we have $rank_j \neq MinRank_j + 1$. As $P_i \in FCand_j \wedge root_i = MaxRoot_j \wedge rank_i = MinRank_j$, the condition of Line 7 is true at P_j and P_j is privileged in Line 9, a contradiction.
  - ◇ Assume that $MaxRoot_j \neq root_j$, *i.e.*, $root_i = root_j < MaxRoot_j$ holds.
    - Suppose that one of the condition of Lines 7, 11 or 15 is true at P_j. Then, P_j is privileged in Lines 8, 12 or 16, a contradiction.
    - Suppose otherwise, *i.e.*, the conditions of Lines 7, 11 and 15 are false at P_j. As $|FCand_j| \geq 1$, the condition of Line 19 is true at P_j. By Line 21, $rank_j = \min\{rank_k \mid P_k \in FCand_j \wedge root_k = root_j\} + 1$ holds. However, as $P_i \in FCand_j \wedge root_j = root_i$, we have $rank_j = \min\{rank_k \mid P_k \in FCand_j \wedge root_k = root_j\} + 1 \leq rank_i + 1 = 1 < rank_j$, a contradiction.

Therefore, the lemma holds. □

As for the previous lemma, in γ', $MaxChildren_i$ is exactly the set of child of process P_i. Hence, according to Definition 5, we have the following lemma:

Lemma 6 *In γ', for each process P_i such that $father_i \neq (D_i, P_i)$ and $MaxChildren_i \geq 3 \vee (MaxChildren_i = 2 \wedge root_i > (D_i, P_i))$ holds, $MaxChildren_i = sdeg_i - 1$ holds.*

Proof First, we observe some properties on the local variables of P_i. As $MaxChildren_i \geq 3 \vee (MaxChildren_i = 2 \wedge root_i > (D_i, P_i) \wedge father_i \neq (D_i, P_i))$, by definition of $FCand_j$, we have:

– $P_i \in FCand_j$ at every $P_j \in N_i$, *i.e.*,
– $|FCand_j| \geq 1$.

If $MaxChildren_i \geq 3$, then as $father_i \neq (D_i, P_i)$, the condition of Line 5 must be false. Thus, $MaxRoot_i \neq (D_i, P_i)$. Consider the case where $MaxChildren_i = 2 \wedge root_i > (D_i, P_i)$. Assume that $MaxRoot_i = (D_i, P_i)$ holds. Then, $root_j \leq MaxRoot_i = (D_i, P_i)$ for every $P_j \in N_i$ by definition of $MaxRoot_i$. However, as $father_i \neq (D_i, P_i)$, by Lines 10, 14, 18 and 22, $father_i = (D_j, P_j)$ and $root_j = root_i$ where $P_j \in N_i$. Thus, $root_i > (D_i, P_i) = MaxRoot_i \geq root_j = root_i$, a contradiction. Therefore, in both cases ($MaxChildren_i \geq 3$ and $MaxChildren_i = 2 \wedge root_i > (D_i, P_i)$), $MaxRoot_i \neq (D_i, P_i)$ holds.

As $rank_j \leq n - 1$ holds for any $P_j \in N_i$ by Lemma 2, $MinRank_i + 1 \leq n$ holds in γ' by definition of $MinRank_i$. Thus, we have $MinRank_i + 1 \leq n$, $MaxChildren_i \geq$

2, and $MaxRoot_i \neq (D_i, P_i)$. Therefore, the condition of Lines 7 or 11 is true. By definition of $MaxRoot_i$, $MaxRoot_i \geq (D_i, P_i) > \emptyset$. By definition of $MaxRoot_j$ for every $P_j \in N_i$, $MaxRoot_j \geq root_i$ holds. Therefore, by Lines 8–9 or 12–13, for every $P_j \in N_i$, we have:

- $MaxRoot_j \geq root_i = MaxRoot_i > (D_i, P_i) > \emptyset$ and
- $rank_i = MinRank_i + 1 \geq 0$.

Based on these observations, we now prove the lemma by the contradiction: Assume that $MaxChildren_i \neq sdeg_i - 1$ for some process P_i such that $father_i \neq (D_i, P_i) \wedge (MaxChildren_i \geq 3 \vee (MaxChildren_i = 2 \wedge root_i > (D_i, P_i)))$ in γ'. By Definition 5, the value of $sdeg_i - 1$ is the number of children if $father_i \neq (D_i, P_i)$. By Line 4, the value of $MaxChildren_i$ is the number of children of P_i plus the size of the set $CCand_i$. Therefore, by assumption $MaxChildren_i \neq sdeg_i - 1$, $|CCand_i| > 0$ holds, which implies that there exists $P_j \in CCand_i$ such that $P_j \in N_i$ and $root_j = \emptyset \vee root_j < MaxRoot_i \vee (root_j = MaxRoot_i \wedge rank_j > MinRank_i + 2)$. Below, we check this condition.

- Case 1: Assume that $root_j = \emptyset$.
  - ◇ Suppose that the condition of Line 5 is true at P_j. Then, P_j is privileged in Line 6 because $root_j \neq (D_j, P_j)$, a contradiction.
  - ◇ Suppose that one of the conditions of Lines 7, 11 or 15 is true. Then, P_j is privileged in Lines 8, 12 or 16 because $MaxRoot_j \neq root_j = \emptyset$, a contradiction.
  - ◇ Otherwise, the condition of Line 19 is true because $|FCand_j| \geq 1$. Then, $\max\{root_k \mid P_k \in FCand_j\} \neq \emptyset$ holds because $root_i \neq \emptyset$ and $P_i \in FCand_j$. Therefore, P_j is privileged in Line 20 because $root_j = \emptyset$, a contradiction.

  Thus, this case does not occur, and we have: $root_j \neq \emptyset$.
- Case 2: Assume that $root_j < MaxRoot_i$.
  - ◇ Suppose that the condition of Line 5 is true at P_j. Then, $MaxRoot_j = (D_j, P_j)$ holds at P_j. However, then P_j is privileged in Line 6 because $root_j < MaxRoot_i = root_i \leq MaxRoot_j = (D_j, P_j)$ which implies $root_j \neq (D_j, P_j)$, a contradiction.
  - ◇ Suppose otherwise that the condition of Line 5 is false at P_j. As $MaxRoot_j \geq root_i = MaxRoot_i$ holds, we have $root_j < MaxRoot_i = root_i \leq MaxRoot_j$, which implies $root_j \neq MaxRoot_j$ and $root_j < root_i$.
    - If one of the conditions of Lines 7, 11 or 15 is true at P_j, P_j is privileged in Lines 8, 12 or 16.
    - Otherwise (*i.e.*, the conditions of Lines 7, 11 and 15 are false), as $P_i \in FCand_j$ and $root_j < root_i$, we have $root_j < root_i \leq \max\{root_k \mid P_k \in FCand_j\}$. Thus, we have $root_j \neq \max\{root_k \mid P_k \in FCand_j\}$ and P_j is privileged in Line 20, a contradiction.
- Case 3: Assume that $root_j = MaxRoot_i \wedge rank_j > MinRank_i + 2$.

  Suppose that the condition of Line 5 is true at P_j. Then, we have $rank_j = 0$ by Line 6. However, this contradicts the assumption $rank_j > MinRank_i + 2 > 0$.

  Suppose otherwise that the condition of Line 5 is false at P_j.
  - ◇ Assume that $MaxRoot_j = root_j$. As $root_j = MaxRoot_j \geq root_i = MaxRoot_i = root_j$ by the assumption, $root_j = MaxRoot_j = root_i = MaxRoot_i$ holds.

- Consider the case where $MinRank_j > rank_i$. As the condition of Line 5 is false at P_j, $MaxChildren_j < 3 \vee MaxRoot_j \neq (D_j, P_j)$.
  - Assume that $MaxRoot_j \neq (D_j, P_j)$ holds. Then, we have $MinRank_j \equiv \min\{rank_k \mid P_k \in N_j \wedge root_k = MaxRoot_j\} > rank_i$. This is a contradiction because $P_i \in N_j \wedge root_i = MaxRoot_j$.
  - Assume that $MaxRoot_j = (D_j, P_j)$ holds. Then, we have $MinRank_j \equiv -1 > rank_i$. This is a contradiction because $rank_i \geq 0$.
- Consider the case where $MinRank_j = rank_i$. As $rank_i = MinRank_i + 1$ holds, $rank_j > MinRank_i + 2 = rank_i + 1 = MinRank_j + 1$ holds, *i.e.*, we have $rank_j \neq MinRank_j + 1$. As $P_i \in FCand_j \wedge root_i = MaxRoot_j \wedge rank_i = MinRank_j$, the condition of Line 7 is true at P_j, and P_j is privileged in Line 9, a contradiction.
- Consider the case where $MinRank_j < rank_i$. By assumption $rank_j > MinRank_i + 2$, we have $MinRank_j < rank_i = MinRank_i + 1 < rank_j - 1$. This implies that $rank_j \neq MinRank_j + 1$ and $rank_i + 1 < rank_j$.
  - Suppose that one of the conditions of Lines 7 or 11 is true at P_j. Then, P_j is privileged in Lines 9 or 13 because $rank_j \neq MinRank_j + 1$, a contradiction.
  - Suppose that the conditions of Lines 7 and 11 are false at P_j. Then, the condition of Line 15 is true at P_j because $P_i \in FCand_j$ and $root_i = MaxRoot_j = root_j$. As $rank_i + 1 < rank_j$, $\min\{rank_k \mid P_k \in FCand_j \wedge root_k = root_j\} + 1 \leq rank_i + 1 < rank_j$. Thus, P_j is privileged in Line 17, a contradiction.

◇ Assume that $MaxRoot_j \neq root_j$.

- Suppose that one of the conditions of Lines 7, 11 or 15 is true at P_j. Then, P_j is privileged in Lines 8, 12 or 16, a contradiction.
- Suppose that the conditions of Lines 7, 11 and 15 are false at P_j. As $|FCand_j| \geq 1$, the condition of Line 19 is true at P_j. By assumption $MaxRoot_i = root_i \leq MaxRoot_j$, $MaxRoot_j \neq root_j$ and $root_j = MaxRoot_i$, we have $root_j = MaxRoot_i = root_i < MaxRoot_j$. Therefore, $root_j = root_i$ holds.

  As $rank_i = MinRank_i + 1$ and $rank_j > MinRank_i + 2$ by assumption, we have $rank_j > rank_i + 1$. By Line 21, $rank_j = \min\{rank_k \mid P_k \in FCand_j \wedge root_k = root_j\} + 1$ holds. However, as $P_i \in FCand_j \wedge root_i = root_j$, we have $rank_j = \min\{rank_k \mid P_k \in FCand_j \wedge root_k = root_j\} + 1 \leq rank_i + 1 < rank_j$, a contradiction.

Therefore, the lemma holds. □

Consider any process P_i such that $father_i \neq (D_i, P_i) \wedge ((MaxChildren_i = 2 \wedge root_i < (D_i, P_i)) \vee MaxChildren_i < 2)$ holds. If $sdeg_i < 1 \vee sdeg_i > 2$ holds, then we can prove that P_i is privileged in Line 4. Hence, follows:

Lemma 7 *In γ', for each process P_i such that $father_i \neq (D_i, P_i) \wedge ((MaxChildren_i = 2 \wedge root_i < (D_i, P_i)) \vee MaxChildren_i < 2)$ holds, $1 \leq sdeg_i \leq 2$ holds.*

Proof Suppose contrary that $father_i \neq (D_i, P_i) \wedge ((MaxChildren_i = 2 \wedge root_i < (D_i, P_i)) \vee MaxChildren_i < 2)$ and $sdeg_i = 0 \vee sdeg_i \geq 3$ holds at P_i in γ' and let derive a contradiction.

As $father_i \neq (D_i, P_i)$, by Lines 10, 14, 18 and 22, $father_i = (D_f, P_f)$ holds with $P_f \in N_i$. Thus, $sdeg_i \geq 1$ holds by Definition 5. Therefore, $sdeg_i \geq 3$, *i.e.*, there exist at least two children P_j and P_k of P_i where $father_j = (D_i, P_i)$ and $father_k = (D_i, P_i)$.

First, suppose that $MaxChildren_i < 2$ holds. As $father_j = (D_i, P_i)$ and $father_k = (D_i, P_i)$, we have $|\{P_j \in N_i \mid father_j = (D_i, P_i)\}| \geq 2$. Thus, P_i is privileged in Line 4 because $MaxChildren_i < 2$, a contradiction.

Next, suppose that $MaxChildren_i = 2 \wedge root_i < (D_i, P_i)$ holds. As $root_i < (D_i, P_i)$, $root_i < (D_i, P_i) \leq MaxRoot_i$ holds by definition of $MaxRoot_i$. By Lines 10, 14, 18 and 22, $root_f = root_i = root_j = root_k$ holds. Then, P_i has at least three neighbors P_f, P_j and P_k such that $root_f = root_j = root_k = root_i < MaxRoot_i$. That is, $\{P_j, P_k, P_f\} \subseteq CCand_i$. Then, P_i is privileged in Line 4 because $MaxChildren_i = 2$, a contradiction. □

The next lemma is deduced from the previous ones:

Lemma 8 *In γ', each tree T represented by the values of fathers is leafy, that is, each T contains at least one process P_l with $sdeg_l \geq 3$, and each process P_i with $sdeg_i = 2$ is adjacent in T to two processes P_j and P_k with $sdeg_j \geq 3$ and $sdeg_k \geq 3$.*

Proof By Lemma 1, each connected component represented by the values of $father_i$ of every process P_i is a tree or a singleton in γ'. Then, by Lemma 1 and Lines 5–6, each tree is rooted at a process P_l with $father_l = root_l = (D_l, P_l)$ and $rank_l = 0$. By the condition of Line 5, the root process P_l satisfies $MaxChildren_l \geq 3$. By Lemma 5, $MaxChildren_l = sdeg_l \geq 3$ because $father_l = (D_l, P_l)$ and $root_l \neq \emptyset$. Therefore, each tree contains at least one process P_l with $sdeg_l \geq 3$.

Now, we observe properties on the local variables of any process P_h such that $1 \leq sdeg_h < 3$. From the above discussion, the condition of Line 5 is false at P_h, and P_h does not execute Line 6. By Lemma 4, if $root_h = \emptyset$ holds, $sdeg_h = 0$ holds and this contradicts the assumption. Thus, we have:

– $root_h \neq \emptyset$.

As $root_h \neq \emptyset$, P_h does not execute Line 24 either. Thus, we have:

– $father_h \neq (D_h, P_h)$.

By Lemmas 6 and 7, because $father_h \neq (D_h, P_h)$ holds, we have:

– $MaxChildren_h < 2 \vee (MaxChildren_h = 2 \wedge root_h < (D_h, P_h))$.

By Lemma 2, $rank_k \leq n - 1$ holds for every $P_k \in N_h$. Thus, by definition of $MinRank_h$, $MinRank_h + 1 \leq n$. Assume that the condition of Line 11 is true at P_h. Then, we have $MaxChildren_h = 2 \wedge root_h < (D_h, P_h)$ and $MaxRoot_h \neq (D_h, P_h)$. By the definition of $MaxRoot_h$, $root_h < (D_h, P_h) < MaxRoot_h$ holds, *i.e.*, $root_h \neq MaxRoot_h$. Thus, P_h is privileged in Line 12, a contradiction. Therefore, P_h executes one of Lines 10, 18 or 22 but not Line 14 to assign the values of $father_h$.

To prove this lemma, we now suppose by contradiction that some process P_i with $sdeg_i = 2$ is adjacent to a process P_j with $sdeg_j < 3$ in T in γ', and we will derive a contradiction.

As P_j is a neighbor of P_i in T, $sdeg_j \geq 1$. Thus, $1 \leq sdeg_j < 3$ holds. Therefore, the above observations on P_h apply to P_i and P_j because $1 \leq sdeg_i = 2 < 3$ and $1 \leq sdeg_j < 3$. There are two cases to consider: P_j is a father or a child of P_i, *i.e.*, $father_i = (D_j, P_j)$ or $father_j = (D_i, P_i)$.

Assume that $father_i = (D_j, P_j)$. Then, as $father_i = (D_j, P_j)$ and by Lines 10, 18 and 22, $P_j \in FCand_i$ holds, *i.e.*, by definition of $FCand_i$, $MaxChildren_j \geq 3 \vee (MaxChildren_j = 2 \wedge root_j > (D_j, P_j) \wedge father_j \neq (D_j, P_j))$ holds. However, this predicate is false by assumption and the above observations for any P_h such that $1 \leq sdeg_h < 3$, a contradiction. That is, P_i such that $1 \leq sdeg_i < 3$ does not select P_j such that $1 \leq sdeg_j < 3$ as its father in γ'.

Assume that $father_j = (D_i, P_i)$. This is a contradiction. Indeed, from the above discussion, P_j, which satisfies $1 \leq sdeg_j < 3$, does not select P_i as its father because $1 \leq sdeg_i = 2 < 3$.

Therefore, the degrees of a father and a child of P_i are greater or equal to 3 in T, *i.e.*, each tree T is leafy in γ'. □

According to the conditions of the algorithm, if we add some processes to a tree in γ', then the tree is no more leafy, that is the forest we obtain is maximal:

Lemma 9 *In γ', the leafy forest F is maximal.*

Proof By Lemmas 1–8, each connected component represented by the values of $father_i$ of every process P_i is a leafy tree or a singleton in γ'.

First, we observe some properties. If P_i is a singleton, by Lemma 4, $sdeg_i = 0$ holds. If P_i is a root of a tree, by Lemma 5, $sdeg_i = MaxChildren_i$ holds. Otherwise, *i.e.*, $father_i \neq (D_i, P_i)$ by Lines 10, 14, 18 and 22, and P_i is said to be "*a non-root process in a tree*". As $father_i \neq (D_i, P_i)$, $sdeg_i \geq 1$ holds. By Lemmas 4–7, $root_i \neq \emptyset$ holds because $sdeg_i \geq 1$.

Suppose that the leafy forest F is not maximal in γ'. That is, there exists two connected components which can be connected to each other in γ'. Assume that P_i and $P_j \in N_i$ belong to different connected components, and the link between P_i and P_j is not in F but can be added to F. That is, P_i can select P_j as its father or P_j can select P_i as its father.

Assume that both of P_i and P_j are singletons. Then $root_i = root_j = \emptyset$ holds by Line 24. By Lemma 4, each singleton has $MaxChildren_i < 3$. However, by Definition 6, each tree must have at least one process such that the degree is greater or equal to 3 in the tree, *i.e.*, there must exist a singleton P_i neighboring to three other singletons in order to make leafy tree only by singletons, then, $MaxChildren_i \geq 3$ must holds by Line 4. This is a contradiction. Therefore, we need not to consider the case where both of P_i and P_j are singletons.

Assume otherwise, that is, P_i or P_j (or both) is in a tree. We consider the case where P_i is in a tree. Then, by Lines 6, 10, 14, 18 and 22, $root_i = root_{father_i}$ holds in γ'. That is, the value of $root_i$ is the name of the tree to which P_i belongs. Then, we have $root_i \neq root_j$ because P_i and P_j belong to different connected components. Assume that, if these two connected components are connected by the link (P_i, P_j), then the new connected component is a leafy tree. Let $sdeg_i$ be the degree of P_i in

the new connected component, and $sdeg'_i$ be the degree of P_i in the old connected components. Then, $sdeg'_i = sdeg_i - 1$. As $sdeg_i$ or $sdeg_j$ (or both) is greater or equal to 3 in γ' by Definition 6, $sdeg'_i \geq 2$ or $sdeg'_j \geq 2$ holds.

– Suppose that P_j is a singleton, *i.e.*, $root_j = \emptyset$ and $father_j = (D_j, P_j)$. Then, $sdeg'_j = 0 \wedge MaxChildren_j < 3$ holds in γ' by Lemma 4. As $sdeg'_j = 0$, $sdeg'_i \geq 2$ holds. That is, P_i has at least one child P_k in the old connected component. As $root_j = \emptyset$, $P_j \in CCand_i$ holds. Therefore, $MaxChildren_i \geq 2$ holds.

  If $father_i = (D_i, P_i)$, the condition of Line 5 is true because P_i is in a tree by Lines 6, 10, 14, 18 or 22, and thus, $MaxChildren_i \geq 3$ holds.

  If $father_i \neq (D_i, P_i)$, $root_{father_i} = root_i = root_k$ holds by Lines 10, 14, 18 and 22. By the definition of $MaxRoot_i$, $(D_i, P_i) \leq MaxRoot_i$ holds. Therefore, if $father_i \neq (D_i, P_i) \wedge root_i < (D_i, P_i)$ hold, $father_i \in CCand_i$ holds. Thus, P_i counts $father_i$, P_j and P_k as $MaxChildren_i$. Then, if $MaxChildren_i = 2 \wedge father_i \neq (D_i, P_i) \wedge root_i < (D_i, P_i)$, P_i is privileged in Line 4, a contradiction.

  Therefore, $(MaxChildren_i = 2 \wedge father_i \neq (D_i, P_i) \wedge root_i > (D_i, P_i)) \vee MaxChildren_i \geq 3$. Thus, by definition of $FCand_j$, $P_i \in FCand_j$ holds, and one of the conditions of Lines 7, 15 or 19 is true at P_j. Then, P_j is privileged in Lines 10, 18 or 22 because $father_j = (D_j, P_j)$ does not represent a neighbor of P_j, a contradiction.

– Assume that P_j is in a tree. Without loss of generality, we assume that $root_i > root_j$ in γ'. By the definition of $MaxRoot_j$, we have $MaxRoot_j \geq root_i$. Therefore, $MaxRoot_j \geq root_i > root_j$ holds, which implies that $MaxRoot_j \neq root_j$.
  ◇ Suppose that P_j is a root of a tree. By Line 6, $root_j = (D_j, P_j)$ holds. Then, $root_j = (D_j, P_j) = MaxRoot_j \geq root_i$ by the condition of Line 5 and the definition of $MaxRoot_j$. This contradicts the assumption $root_i > root_j$.
  ◇ Consider the case where P_j is a non-root process in a tree.

    If one of the conditions of Lines 7, 11 or 15 is true at P_j, P_j is privileged in Lines 8, 12 or 16 because $MaxRoot_j \neq root_j$, a contradiction.

    Consider the case where the conditions of Lines 7, 11 and 15 are false at P_j.
    • Consider the case where $sdeg'_i \geq 2$.

      By Lines 6, 8, 12, 16, 20 and 24, $MaxRoot_i \geq root_i > root_j$ holds because $root_i > root_j$. By definition of $CCand_i$, $P_j \in CCand_i$ holds. As $sdeg'_i \geq 2$ and $father_j \neq (D_i, P_i)$, if $father_i = (D_i, P_i)$, then P_i has at least two children in the old tree, and $MaxChildren_i \geq 3$ holds. Otherwise, *i.e.*, $father_i \neq (D_i, P_i)$, then P_i has at least one child on the old tree, and $MaxChildren_i \geq 2$ holds.

      Suppose that $MaxChildren_i = 2 \wedge father_i \neq (D_i, P_i) \wedge root_i < (D_i, P_i)$ holds. Then, we have $root_j < root_k = root_{father_i} = root_i < (D_i, P_i) \leq MaxRoot_i$ where P_k is a child of P_i in the old tree by definition of $MaxRoot_i$. Then, $\{P_j, P_k, father_i\} \subseteq CCand_i$ holds, and P_i is privileged in Line 4 because $MaxChildren_i = 2$, a contradiction.

      Therefore, $MaxChildren_i \geq 3 \vee (MaxChildren_i = 2 \wedge father_i \neq (D_i, P_i) \wedge root_i > (D_i, P_i))$ holds. As $rank_i + 1 \leq n$ by Lemma 2, $P_i \in FCand_j$ holds.
      ○ Suppose that $MaxRoot_j = root_i$ holds. Then, the conditions of Lines 7 or 15 are true at P_j because $P_i \in FCand_j$, a contradiction.
      ○ Suppose that $MaxRoot_j > root_i$ holds. Then, P_j is privileged in Line 20 because $P_i \in FCand_j$ and $root_j \neq \max\{root_k \mid P_k \in FCand_j\} \geq root_i > root_j$ hold, a contradiction.

- We consider otherwise, *i.e.*, $sdeg'_j \geq 2 \wedge sdeg'_i < 2$. As $sdeg'_j \geq 2$ and $father_j \neq (D_j, P_j)$, P_j has at least one child P_k. By Lines 10, 14, 18 and 22, $root_{father_j} = root_j = root_k$ holds. As $MaxRoot_j \geq root_i > root_j = root_{father_j} = root_k$, $\{father_j, P_k\} \subseteq CCand_j$ by definition of $CCand_j$, *i.e.*, $MaxChildren_j \geq 2$ by Line 4. By Lemma 2, $rank_l + 1 \leq n$ for each process P_l, *i.e.*, $MinRank_j + 1 \leq n$. As the condition of Line 11 is false, $MaxChildren_j < 2 \vee MaxRoot_j = (D_j, P_j)$ holds. Therefore, $MaxRoot_j = (D_j, P_j)$ holds because $MaxChildren_j \geq 2$. Then, we have $root_{father_j} = root_j = root_k < root_i \leq MaxRoot_j = (D_j, P_j)$.
  - Suppose that $root_i = MaxRoot_j = (D_j, P_j)$ holds. Then, $root_j$ must be (D_j, P_j) because the value of $root_i$ is a copy of the value of a neighbor by Lines 8, 12, 16, 20, except the case where $root_i = (D_i, P_i)$ by Line 6. This is a contradiction because $root_i \neq root_j$.
  - Suppose that $root_i < MaxRoot_j = (D_j, P_j)$ holds. Then, we have $\{P_i\} \subseteq CCand_j$. Then, $MaxChildren_j \geq 3$ holds, and the condition of Line 5 is true. This is a contradiction because P_j is a non-root process.

Therefore, the leafy forest F is maximal in γ'. □

By Lemmas 1–9, we have the following lemma:

Lemma 10 *The configuration γ' is legitimate.*

Proof In γ', each connected component T represented by the value of $father_i$ is a tree or a singleton in G (Lemma 1). 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 (Lemma 2).

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 (Lemmas 3 and 5). Additionally, each singleton P_i has $father_i = (D_i, P_i)$, $root_i = \emptyset$, and $sdeg_i = 0$ (Lemma 4). 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 (Lemmas 6, 7 and 8), and the leafy forest is maximal (Lemma 9). □

The second part of the proof consists in proving that every computation of SSMLF eventually reaches a terminal configuration.

We prove that, for any configuration γ and any computation starting from γ, each value of $root_i$ of each process P_i in G eventually become less or equal to MAX, and P_0 such that $MAX = (D_0, P_0)$ decides the value of its variables as a root of a tree and never changes (Lemma 11). After that, some processes P_i eventually form the tree $T_0 = (V_0, E_0)$ rooted at P_0 and never change the value of their variables (Lemma 15).

Let $G^1 = (V^1, E^1)$ be an induced subgraph of G by $V \setminus V_0$, and $(D_{\bar{1}}, P_{\bar{1}})$ be the maximum pair in processes in G^1 that are not neighboring of any process in V_0. If $D_{\bar{1}} \geq 3$, $P_{\bar{1}}$ eventually decides the value of its variables as a root and never changes (Lemma 14). After that, some processes P_i eventually form the tree $T_1 = (V_1, E_1)$ rooted at $P_{\bar{1}}$ and never change the value of their variables (Lemma 15).

By repeating this discussion, we get a series of trees $T_0 = (V_0, E_0)$, $T_1 = (V_1, E_1), \ldots, T_k = (V_k, E_k)$, where each process in $V_0, V_1, \ldots, V_k$ fixes the value of its each variable, and eventually no process is privileged. Now let start proving each lemma.

By Lines 1–2 and 5–6 of the algorithm, we have:

Lemma 11 *For any configuration γ and any computation starting from γ, the value of $root_i$ of each process P_i in G eventually becomes less or equal to MAX, and P_0 such that $MAX = (D_0, P_0)$ decides the value of its variables as a root of a tree*, i.e. $root_0 = father_0 = (D_0, P_0)$ *and* $rank_0 = 0$.

Proof By Lines 1–2, it is clear that $root_i \leq MAX$ holds for each process P_i. By Lines 3–4, $MaxChildren_0 = D_0 \geq 3$ eventually holds. Then, as $MaxChildren_0 \geq 3 \wedge MaxRoot_0 = MAX$, the condition of Line 5 is true at P_0. Then, P_0 eventually becomes a root by Line 6; $root_0 = father_0 = MAX$, $rank_0 = 0$, and P_0 does not change the value of each variable in the following steps by definition of the algorithm. □

Definition 9 For each k (> 0), let γ_k be a configuration such that, in any computation starting from γ_k:

- $T_t = (V_t, E_t)$ $(0 \leq t < k)$ is a leafy tree, and
- each process in V_t $(0 \leq t < k)$ never changes the value of its variables in any configuration in the computation.

For each k, let $G^k = (V^k, E^k)$ be an induced subgraph of G by $V \setminus \{V_0 \cup V_1 \cup \cdots \cup V_{k-1}\}$.

Let $P_i \in V^k$ be a process that is neighbor of some process in $V \setminus V^k$. As P_i does not join any tree, the conditions of Lines 5, 7, 11 and 15 are false. Hence, we have the following lemma:

Lemma 12 *For each k and any computation starting from γ_k, each process $P_i \in V^k$ which is neighbor of some process in $V \setminus V^k$ does not become a root of a tree.*

Proof Consider each process $P_i \in V^k$ which is neighbor of any process in $V \setminus V^k$. As P_i is neighbor of some trees $T_0, T_1, \ldots, T_{k-1}$, we have $MaxRoot_i = (D_l, P_l) > (D_i, P_i)$, where P_l is one of the roots of trees $T_0, T_1, \ldots, T_{k-1}$. If one of the conditions of Lines 5, 7, 11 or 15 is true, P_i joins the tree rooted at $MaxRoot_i$. As P_i does not join any trees, these conditions are false. As the condition of Line 11 is false, $MaxChildren_i < 2$ holds. Therefore, by Line 5, P_i does not become a root of a tree because of $MaxChildren_i < 2$. □

By Lemma 12, we consider the processes that are not neighbor of any process in $V \setminus V^k$ as candidates of roots of leafy trees.

Definition 10 For each k (> 0), let $(D_{\bar{k}}, P_{\bar{k}})$ be the maximum pair in processes in V^k that are not neighbor of any process in $V \setminus V^k$, *i.e.*, $(D_{\bar{k}}, P_{\bar{k}}) = \max\{(D_j, P_j) \mid P_j \in V^k$ and P_j is not a neighbor of any process in $V \setminus V^k\}$.

If there exists a process P_i in G^k such that $root_i$ is larger than $(D_{\bar{k}}, P_{\bar{k}})$ in any configuration in the computation, then such P_i eventually executes Line 24, and $root_i < (D_{\bar{k}}, P_{\bar{k}})$ holds. Hence, we have the following lemma:

Lemma 13 *For each k and any computation starting from γ_k, the value of $root_i$ of each process P_i in G^k eventually becomes less or equal to $(D_{\bar{k}}, P_{\bar{k}})$.*

Proof Let $L \subset V^k$ be a set of processes P_i such that $root_i$ is larger than $(D_{\bar{k}}, P_{\bar{k}})$ in any configuration in the computation. If each $P_i \in L$ executes Line 6, we have $root_i = (D_i, P_i) < (D_{\bar{k}}, P_{\bar{k}})$ because, by Lemma 12, P_i is not neighboring to any process in $V \setminus V^k$. If each $P_i \in L$ executes Line 24, we have $root_i = \emptyset < (D_{\bar{k}}, P_{\bar{k}})$. Therefore, each $P_i \in L$ never executes Lines 6 and 24, but P_i executes Lines 8–10, 12–14, 16–18 or 20–22 to change the value of $root_i$, $rank_i$ and $father_i$. By execution of these lines, each $P_i \in L$ satisfies $root_i = root_{father_i}$, $rank_i = rank_{father_i} + 1$, and $father_i \in N_i$.

Therefore, there exists no process P_i such that $father_i = (D_i, P_i)$ in L. For each $P_i \in L$, there exists a loop $P_0, P_1, \ldots, P_i, \ldots, P_{m-1}, P_0$ whose length is $m \geq 2$ such that $father_0 = (D_1, P_1), father_1 = (D_2, P_2), \ldots, father_{m-1} = (D_0, P_0)$ by Lines 10, 14, 18 or 22. However, there exists a process P_l such that $rank_l \neq rank_{father_l} + 1$ in the loop, and P_l is privileged in Lines 9, 13, 17 or 21. Then, P_l executes one of these lines, and the value of $rank_l$ increases. Thus, as the computation continues, for each $P_i \in L$, the value of $rank_i$ eventually becomes greater or equal to n. Then, the conditions of Lines 7, 11, 15 and 19 become false at P_i, P_i executes Line 24, and we have $root_i \leq (D_{\bar{k}}, P_{\bar{k}})$. After that, every $P_i \in L$ eventually holds $root_i \leq (D_{\bar{k}}, P_{\bar{k}})$, and L eventually becomes an empty set.

Therefore, each value of $root_i$ in G^k eventually becomes less or equal to the maximum $(D_{\bar{k}}, P_{\bar{k}})$ in G^k, and $root_i \leq (D_{\bar{k}}, P_{\bar{k}})$ is maintained forever. □

As $MaxChildren_{\bar{k}} \geq 3$ eventually holds and remains so forever, in the condition of Line 5, $MaxRoot_{\bar{k}} = (D_{\bar{k}}, P_{\bar{k}}) \wedge MaxChildren_{\bar{k}} \geq 3$ eventually holds and remains so forever. Hence, we have:

Lemma 14 *For each k and any computation starting from γ_k, if $D_{\bar{k}} \geq 3$, $P_{\bar{k}}$ eventually decides the value of its variables as a root.*

Proof First, suppose that $P_{\bar{k}}$ does not become a root forever. By Lemma 13, for any neighbor P_i of $P_{\bar{k}}$, $root_i \leq (D_{\bar{k}}, P_{\bar{k}})$ eventually holds. As $D_{\bar{k}} \geq 3$, there exist at least 3 neighbors of $P_{\bar{k}}$, such as P_i, in G^k. By definition of $MaxRoot_{\bar{k}}$, $MaxRoot_{\bar{k}} = (D_{\bar{k}}, P_{\bar{k}})$ holds, and by the definition of $MinRank_{\bar{k}}$, $MinRank_{\bar{k}} = -1$ holds. Then, because $P_{\bar{k}}$ does not become a root by assumption, $MaxChildren_{\bar{k}} < 3$ holds forever by the condition of Line 5. Thus, by Lines 3–4, $P_i \notin CCand_{\bar{k}} \wedge father_i \neq (D_{\bar{k}}, P_{\bar{k}})$ must hold. Putting things together and by definition of $CCand_{\bar{k}}$, $root_i = MaxRoot_{\bar{k}} = (D_{\bar{k}}, P_{\bar{k}}) \wedge rank_i \leq 1 \wedge father_i \neq (D_{\bar{k}}, P_{\bar{k}})$ holds.

As $root_i = (D_{\bar{k}}, P_{\bar{k}})$ hold, P_i does not execute Lines 6 and 24. If P_i executes Line 6, we have $root_i = (D_i, P_i) < (D_{\bar{k}}, P_{\bar{k}})$. If P_i executes Line 24, we have $root_i = \emptyset < (D_{\bar{k}}, P_{\bar{k}})$. Therefore, P_i executes Lines 8–10, 12–14, 16–18 or

20–22 to change the value of $root_i$, $rank_i$ and $father_i$. By executing these lines, $root_i = root_{father_i} = (D_{\bar{k}}, P_{\bar{k}})$, $rank_i = rank_{father_i} + 1$, and $father_i \in N_i$ hold. As $root_j = root_{father_j}$ for each process P_j, we consider the set of processes with $root_j = (D_{\bar{k}}, P_{\bar{k}})$. Let L be such a set. As $P_{\bar{k}}$ does not become a root by assumption, there eventually exists no process P_j with $father_j = (D_j, P_j)$ in L. Therefore, there exists a loop $P_0, P_1, \ldots, P_i, \ldots, P_{m-1}, P_0$ whose length is $m \geq 2$ such that $father_0 = (D_1, P_1), father_1 = (D_2, P_2), \ldots, father_{m-1} = (D_0, P_0)$ in L by Lines 10, 14, 18 or 22. Then, there exists a process P_l such that $rank_l \neq rank_{father_l} + 1$ in the loop, and P_l is privileged in Lines 9, 13, 17 or 21. Then, P_l executes one of these lines, and the value of $rank_l$ increases. Then, as the computation continues, each value of $rank_i$ eventually becomes larger than 1 by Lines 9, 13, 17 or 21. Then, because $root_i = (D_{\bar{k}}, P_{\bar{k}}) = MaxRoot_{\bar{k}}$ and $rank_i > MinRank_{\bar{k}} + 2 = -1 + 2 = 1$ holds, *i.e.* $P_i \in CCand_{\bar{k}}$ holds, P_i is eventually counted in $MaxChildren_{\bar{k}}$. As there are three such processes P_i, $MaxChildren_{\bar{k}}$ eventually becomes greater or equal to 3. Then, because the condition of Line 5 becomes true, $P_{\bar{k}}$ becomes a root by Line 6.

Next, suppose that $P_{\bar{k}}$ does not fix the value of its variables as a root forever in G^k. This case implies that, $P_{\bar{k}}$ repeats becoming and resigning a root of a tree alternately forever. When $P_{\bar{k}}$ becomes a root, by the condition of Line 5, $P_{\bar{k}}$ must satisfy $MaxChildren_{\bar{k}} \geq 3 \wedge MaxRoot_{\bar{k}} = (D_{\bar{k}}, P_{\bar{k}})$. When $P_{\bar{k}}$ resigns the root, then $MaxChildren_{\bar{k}} < 3 \vee MaxRoot_{\bar{k}} \neq (D_{\bar{k}}, P_{\bar{k}})$ must hold. As $(D_{\bar{k}}, P_{\bar{k}})$ is the maximum pair, the value of $root_{\bar{k}}$ is the maximum value in G^k. Therefore, $MaxRoot_{\bar{k}} = (D_{\bar{k}}, P_{\bar{k}})$ does not change forever. That is, $MaxChildren_{\bar{k}} \geq 3$ and $MaxChildren_{\bar{k}} < 3$ must hold alternately forever so that $P_{\bar{k}}$ repeats becoming and resigning a root.

By definition of $MaxRoot_i$ for any $P_i \in N_{\bar{k}}$ and by Lemma 13, $root_i \leq MaxRoot_i = root_{\bar{k}} = (D_{\bar{k}}, P_{\bar{k}}) = MaxRoot_{\bar{k}}$ holds, and this relation is maintained forever.

- If $root_i < MaxRoot_i$, then $root_i < MaxRoot_{\bar{k}}$ holds, *i.e.*, $P_i \in CCand_{\bar{k}}$ holds.
- If $root_i = MaxRoot_i$, then $root_i = root_{\bar{k}} = MaxRoot_i$ holds.
  - ◇ If $father_i = (D_{\bar{k}}, P_{\bar{k}})$, then $P_i \in \{P_j \in N_{\bar{k}} \mid father_j = (D_{\bar{k}}, P_{\bar{k}})\}$ holds.
  - ◇ If $father_i = (D_j, P_j) \neq (D_{\bar{k}}, P_{\bar{k}})$, then $root_j = root_i = MaxRoot_i = root_{\bar{k}} = (D_{\bar{k}}, P_{\bar{k}})$. However, $rank_j > 0$ holds by Lines 9, 13, 17 and 21 because $root_j \neq (D_j, P_j)$. Then, $rank_i = rank_j + 1 > 1$ holds. Thus, $rank_i > MinRank_{\bar{k}} + 2 = 1$ holds, *i.e.*, $P_i \in CCand_{\bar{k}}$ holds.

Therefore, in any case, because P_i is counted in $CountMaxChildren_{\bar{k}}$ by $P_{\bar{k}}$, P_i is counted in $MaxChildren_{\bar{k}}$ by $P_{\bar{k}}$ in Line 4. As $D_{\bar{k}} \geq 3$, there exists at least three such processes, and $MaxChildren_{\bar{k}} \geq 3$ holds and remains so forever. Thus, $P_{\bar{k}}$ eventually stops resigning a root of a tree. Therefore, $P_{\bar{k}}$ eventually decides the value of its variables as a root. □

By Lemmas 13 and 14, any root of trees can be discussed by the same way as the root P_0 such that $MAX = (D_0, P_0)$ in the following lemma. In ascending order of the value of $rank_i$ from $rank_{\bar{k}} = 0$ of the root $P_{\bar{k}}$, each process P_i fixes the value of each variable if it joins T_k.

Lemma 15 *For each k and any computation starting from γ_k, let γ_p be a configuration such that $P_{\bar{k}}$ decides the value of its variables as a root of a tree in G^k. Then,*

for any computation starting from γ_p, some processes eventually form a leafy tree $T_k = (V_k, E_k)$ rooted at $P_{\bar{k}}$ and never change the value of their variables.

Proof We prove that, for any computation starting from γ_p, in ascending order of the value of *rank* from the root $P_{\bar{k}}$, each process fixes the value of each variable if it joins T_k.

Suppose that there exists a computation c in which there exists a process in G^k which repeatedly joins and leaves T_k forever. Let P_i be such a process which is the nearest from $P_{\bar{k}}$ in T_k, *i.e.*, the value of $rank_i$ is the smallest value when P_i is in T_k. Let $h(>0)$ be the value of $rank_i$.

- Let Z be a set of processes P_z such that the value of its variables $rank_z < h-1$, $father_z$, $MaxChildren_z$ and $root_z = (D_{\bar{k}}, P_{\bar{k}})$ are fixed in c.
- Let J be a set of processes P_j such that the value of its variables $rank_j = h-1$, $father_j$ and $root_j = (D_{\bar{k}}, P_{\bar{k}})$ are fixed, and $MaxChildren_j \geq 2$ or $MaxChildren_j < 2$ are not fixed.
- Let W be a set of processes $P_w \in V^k \setminus \{Z \cup J\}$ such that the value of its variables are changed forever.
- Let $I \subseteq W$ be a set of processes which are neighboring to any member of J. Then, $P_i \in I$ holds.

The value of $MaxChildren_j$ where $P_j \in J$ is changed based on the value of $father_i$ of $P_i \in I$ in Line 4. As $P_j \in N_i$, $MaxRoot_i = (D_{\bar{k}}, P_{\bar{k}})$ holds by definition of $MaxRoot_i$, and $MinRank_i = h-1$ holds by definition of $MinRank_i$.

Consider the case where $h = 1$. Then, because $P_{\bar{k}}$ never change its value, the root process $P_{\bar{k}}$ is in J. By definition of $MaxRoot_i$, $MinRank_i$ and $FCand_i$, $MaxRoot_i = (D_{\bar{k}}, P_{\bar{k}})(> (D_i, P_i))$, $MinRank_i = 0$ and $P_{\bar{k}} \in FCand_i$ hold. Therefore, the condition of Line 7 is true at P_i, and P_i executes Lines 8–10; $root_i = father_i = (D_{\bar{k}}, P_{\bar{k}})$, and $rank_i = 1$ hold. After that, P_i does not change the value of its these variables by definition of the algorithm. Therefore, this case does not occur.

Consider the case where $h \geq 2$. If P_i joins T_k, by Lines 10, 14, 18 and 22, $root_i = (D_{\bar{k}}, P_{\bar{k}})$, $rank_i = h$ and $father_i$ represents one of its neighbors $N_i \cap J$. Then, the condition for which P_i joins T_k is the following by Lines 7, 11 and 15: $MaxChildren_i \geq 2 \vee \exists P_j \in J \cap N_i, [P_j \in FCand_i]$. The condition for which P_i leaves T_k is following: $MaxChildren_i < 2 \wedge \forall P_j \in J \cap N_i, [P_j \notin FCand_i]$. Thus, $MaxChildren_i$ or $FCand_i$ must change their values forever. By definition of $FCand_i$, $MaxChildren_j$ must change its value forever, since $father_j$ and $root_j = (D_{\bar{k}}, P_{\bar{k}})$ are fixed by the assumption. Therefore, $MaxChildren_i$ or $MaxChildren_j$ must change their values forever. That is, $MaxChildren_j \geq 2$ and $MaxChildren_j < 2$ hold at P_j alternately forever or $MaxChildren_i \geq 2$ and $MaxChildren_i < 2$ hold at P_i alternately forever.

First, we suppose that $MaxChildren_j \geq 2$ and $MaxChildren_j < 2$ hold alternately forever. Let (D_l, P_l) be the largest pair among processes in J with $MaxChildren_l \geq 2$. Then, for each $P_m \in I \cap N_l$, $P_l \in FCand_m$ holds. As $root_l = MaxRoot_m = (D_{\bar{k}}, P_{\bar{k}})$ and $rank_l = MinRank_m = h-1$ hold for each $P_m \in I \cap N_l$ and (D_l, P_l) is the largest pair, by Line 10, P_m selects P_l as its father. Then, $MaxChildren_l \geq 2$ is maintained forever, and P_l eventually leaves the set J. Therefore, P_j eventually leaves J and decides $MaxChildren_j \geq 2$ or $MaxChildren_j < 2$.

Next, we suppose that $MaxChildren_i \geq 2$ and $MaxChildren_i < 2$ hold alternately forever. For each $P_a \in I$, if $MaxChildren_a \geq 2$ and $MaxChildren_a < 2$ hold alternately, by definition of $MaxChildren_a$ and $CCand_a$, there exists a process $P_{a'} \in N_a$ such that $root_{a'} = \emptyset \vee root_{a'} < (D_{\bar{k}}, P_{\bar{k}}) \vee \{root_{a'} = (D_{\bar{k}}, P_{\bar{k}}) \wedge (rank_{a'} > h + 1 \vee (rank_{a'} = h + 1 \wedge father_{a'} = (D_a, P_a)))\}$ (*i.e.*, $P_{a'}$ is a child or a child-candidate of P_a) and $root_{a'} = (D_{\bar{k}}, P_{\bar{k}}) \wedge rank_{a'} = h + 1 \wedge father_{a'} \neq (D_a, P_a)$ (*i.e.*, $P_{a'}$ becomes a child of a process that is not P_a, we call such event "P_a is bereaved its children and child-candidates by other process") hold alternately. When the latter holds, there exists $P_b \in N_{a'} \cap I$ such that P_b bereaves $P_{a'}$ from P_a. Then, $P_b \in FCand_{a'}$ and $(D_b, P_b) > (D_a, P_a)$ hold by the condition of Line 7 and Lines 8–10. When the former holds, there exists no such process in $N_{a'} \cap I$, *i.e.* $P_b \notin FCand_{a'}$. That is, $P_b \in FCand_{a'}$ and $P_b \notin FCand_{a'}$ must hold alternately, *i.e.*, $MaxChildren_b \geq 2$ and $MaxChildren_b < 2$ must hold alternately by definition of $FCand_{a'}$. Then, there must exists a process P_c by which P_b is bereaved its children and child-candidates $P_{b'}$, and $(D_c, P_c) > (D_b, P_b)$ holds. Then, $MaxChildren_c \geq 2$ and $MaxChildren_c < 2$ must hold alternately. By following such a relation, we have a list of processes $P_a, P_{a'}, P_b, P_{b'}, P_c, \ldots, P_{(x-1)}, P_{(x-1)'}, P_x$ such that (D_x, P_x) is the largest pair in I, and $MaxChildren_x \geq 2$ and $MaxChildren_x < 2$ hold alternately forever. When $MaxChildren_x \geq 2$, there exists no process which bereaves the children and child-candidates of P_x because (D_x, P_x) is the largest. Thus, its children $P_{(x-1)'}$ never change the value of $father_{(x-1)'}$, and $MaxChildren_x \geq 2$ is maintained forever. This is a contradiction. Therefore, $MaxChildren_i \geq 2$ or $MaxChildren_i < 2$ is eventually fixed.

Therefore, P_i in T_k does not change the value of its variables, then, the members of T_k do not change forever. □

By Lemmas 11–15, we have the following lemma.

Lemma 16 *For any configuration γ and any computation starting from γ, eventually no process is privileged.*

Proof For any configuration γ and any computation starting from γ, each value of $root_i$ of each process P_i in G eventually become less or equal to MAX, and P_0 such that $MAX = (D_0, P_0)$ decides the value of its variables as a root of a tree and never changes (Lemma 11). After that, some processes P_i eventually form the tree $T_0 = (V_0, E_0)$ rooted at P_0 and never change the value of their variables (Lemma 15).

Consider $G^1 = (V^1, E^1)$ and $(D_{\bar{1}}, P_{\bar{1}})$. If $D_{\bar{1}} \geq 3$, $P_{\bar{1}}$ eventually decides the value of its variable as a root and never changes (Lemma 14). After that, some processes P_i eventually form the tree $T_1 = (V_1, E_1)$ rooted at $P_{\bar{1}}$ and never change the value of their variable (Lemma 15).

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

By Lemmas 10 and 16, we have the following theorem.

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

We now analyze the time complexity of our algorithm in terms of *rounds*. We define a round as a minimal period in which each privileged process P_i at the beginning either executes once or becomes non-privileged. In real distributed networks, it is more natural to evaluate the time complexity in terms of rounds because this notion captures the execution rate of the slowest process in any execution.

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

Proof In the first round, each process P_i with $root_i > MAX$ executes Line 2, and the process P_0 such that $MAX = (D_0, P_0)$ executes Line 4 and $MaxChildren_0 = D_0 \geq 3$ holds. In the second round, P_0 executes Line 6 and becomes a root by Lemma 11. In the third round, the neighbors P_i of P_0 execute Lines 8–10 and fix the value of $father_i$, $root_i$ and $rank_i$ (see the case that $h = 1$ in the proof of Lemma 15).

According to the proof of Lemma 15, we consider the time which are needed to construct a tree T_0 rooted at P_0. Note that h is the distance from P_0 on T_0. From the forth round, the neighbors P_i of P_0, *i.e.* processes in the case of $h = 1$, decides the value of $MaxChildren_i$. After that, processes in the case of $h = 2$ decides the value of their valuables. In the ascending order of h, each process fixes the value of its valuables. Additionally, in each h, each process executes Line 4 to decide the value of $MaxChildren_i$, and executes Lines 8–10, 12–14, 16–19, 20–22 or 24 to decide the value of $father_i$, $root_i$ and $rank_i$. In each round, at least one process P_i decides the value of $MaxChildren_i$ or the value of its other variables. Therefore, the tree T_0 rooted at P_0 are constructed by $O(n)$ rounds.

After that, by the proof of Lemma 13, in each round, each process P_i with $root_i > (D_{\bar{1}}, P_{\bar{1}})$ counts up the value of $rank_i$ by at least 1 to n. After each value of $rank_i$ becomes greater or equal to n, in each round, at least one process with $root_i > (D_{\bar{1}}, P_{\bar{1}})$ executes Line 24. Therefore, such reset of the value of $root_i$ as mentioned in Lemma 13 needs $O(n)$ rounds. At the same time, each value of $rank_i$ of each process with $root_i = (D_{\bar{1}}, P_{\bar{1}})$ becomes greater than 1 if $P_{\bar{1}}$ is not a root. After that, in the next round, $P_{\bar{1}}$ executes Line 4 and Line 6 and becomes a root by Lemma 14.

The execution of algorithm repeats building a tree T_k and reset the value of *root* in $V \setminus \{V_0 \cup V_1 \cup \cdots \cup V_k\}$. Each building of a tree needs $O(n)$ rounds, and each reset needs $O(n)$ rounds. Since the algorithm builds at most n trees, the total time complexity of SSMLF is $O(n^2)$ rounds. □

3.2 The third layer: assignment of edge cost

Formal description of the third layer SSTN is shown in Fig. 4. This layer computes a network cost from the MLF computed by the second layer. In the fourth layer, the minimum cost spanning tree is computed based on this cost. The minimum cost spanning tree is the approximate solution of the MLST problem. That is, all *tree* edges

Constants (Inputs)
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 (an output from the second layer).
$father_i$: ID of the father of P_i in T (an output from the second layer).
Variables (Outputs)
$W(P_i)[P_j]$: 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. 4 SSTN: Self-Stabilizing algorithm for Transforming the Network

of the MLF must be tree-edges of the minimum cost spanning tree, and all *intra-tree* edges, that is non-tree-edges having both endpoints in the same tree of the MLF, must not be in the minimum cost 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, that is non-tree-edges having their endpoints not in the same tree.

In SSTN, the inputs of each process P_i are:

– $root_i$: ID of the root of leafy tree T to which P_i belongs, and
– $father_i$: ID of the father of P_i in T.

Based on these inputs, the output of each process P_i is the cost $W(P_i)[P_j]$ of each edge between P_i and its neighbors P_j.

Definition 11 A configuration of SSTN is legitimate if and only if each process satisfies the following three conditions for each edge e between P_i and its neighbors P_j.

– 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.

We now prove the correctness of SSTN.

Lemma 17 *Let γ' be a configuration. No process is privileged in γ' if and only if γ' is legitimate.*

Proof Consider any illegitimate configuration. Let $e = (P_i, P_j)$ be an edge.

– Suppose that e is a tree edge, but $W(P_i)[P_j]$ is not 0. Then, $W(P_i)[P_j] \neq 0$, $root_i = root_j$, and $father_i = P_j \vee father_j = P_i$. Thus, P_i is privileged in Lines 5–6.

– Suppose that e is an intra-tree edge, but $W(P_i)[P_j]$ is not ∞. Then, $W(P_i)[P_j] \neq \infty$, $root_i = root_j$, and $father_i \neq P_j \wedge father_j \neq P_i$. Thus, P_i is privileged in Lines 3–4.
– Suppose that e is an inter-tree edge, but $W(P_i)[P_j]$ is not 1. Then, $W(P_i)[P_j] \neq 1$ and $root_i \neq root_j$. Thus, P_i is privileged in Lines 1–2.

Therefore, at least one process is privileged in illegitimate configuration. Moreover, it is clear that no process is privileged if the configuration is legitimate. □

Lemma 18 *For any configuration γ and any computation starting from γ, eventually no process is privileged.*

Proof Let $e = (P_i, P_j)$ be any edge. P_i executes Lines 2, 4 or 6 for e at most once. Thus, there is no infinite computation. □

By Lemmas 17 and 18, we have the following theorem:

Theorem 4 *The algorithm* SSTN *is self-stabilizing with respect to Λ_t.*

Theorem 5 *The time complexity of algorithm* SSTN *is $O(n)$ rounds.*

Proof Each process executes at most once for each incident edge. Therefore, each process P_i executes at most δ_i rounds where δ_i is the degree of P_i. Time complexity of this algorithm is then bounded by $O(\max_i \delta_i) = O(n)$. □

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

Theorem 6 *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 networks. Its approximation ratio is 3. However, there exists a sequential solution proposed by Solis-Oba (1998) 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.

Additionally, we assume a distributed weakly fair daemon as its scheduler. For other kinds of scheduler, we will consider designing algorithms in future research.

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. In addition to being self-stabilizing, a protocol could thus also tolerate a limited number of topology changes (Dolev and Herman 1997), crash faults (Gopal and Perry 1993; Anagnostou and Hadzilacos 1993), nap faults (Dolev and Welch 1997; Papatriantafilou and Tsigas 1997), Byzantine faults (Dolev and

Welch 2004; Ben-Or et al. 2008), and sustained edge cost changes (Cobb and Gouda 2002; Johnen and Tixeuil 2003). Investigating the possibility to add such properties to our MLST protocol is an intriguing open question.

Acknowledgements This work is supported in part by a Grant-in-Aid for Young Scientists ((B)22700-074) of JSPS, Kayamori Foundation of Informational Science Advancement, a Grant-in-Aid for Scientific Research ((B)20300012) of JSPS, "Global COE (Centers of Excellence) Program" of the Ministry of Education, Culture, Sports, Science and Technology, Japan, and ANR projects SHAMAN, ALADDIN, and SPADES.

References

Anagnostou E, Hadzilacos V (1993) Tolerating transient and permanent failures (extended abstract). In: Proceedings of the 7th international workshop on distributed algorithms. LNCS, vol 725. Springer, Berlin, pp 174–188

Ben-Or M, Dolev D, Hoch EN (2008) Fast self-stabilizing byzantine tolerant digital clock synchronization. In: Proceedings of the twenty-seventh ACM symposium on principles of distributed computing, pp 385–394

Blin L, Potop-Butucaru M, Rovedakis S (2009) Self-stabilizing minimum-degree spanning tree within one from the optimal degree. In: Proceedings of the 23th IEEE international parallel and distributed processing symposium.

Blin L, Potop-Butucaru MG, Rovedakis S, Tixeuil S (2009) A new self-stabilizing minimum spanning tree construction with loop-free property. In: Proceedings of the 23rd international symposium on distributed computing. LNCS, vol 5805. Springer, Berlin, pp 407–422

Butelle F, Lavault C, Bui M (1995) A uniform self-stabilizing minimum diameter tree algorithm. In: Proceedings of the 9th international workshop on distributed algorithms. LNCS, vol 972. Springer, Berlin, pp 257–272

Cobb JA, Gouda MG (2002) Stabilization of general loop-free routing. J Parallel Distrib Comput 62:922–944

Datta AK, Larmore LL, Vemula P (2008) Self-stabilizing leader election in optimal space. In: Proceedings of the 10th international symposium on stabilization, safety, and security of distributed systems. LNCS, vol 5340. Springer, Berlin, pp 109–123

Dijkstra EW (1974) Self-stabilizing systems in spite of distributed control. Commun ACM 17(11):643–644

Dolev S (2000) Self-stabilization. MIT Press, Cambridge

Dolev S, Herman T (1997) Superstabilizing protocols for dynamic distributed systems. Chic J Theor Comput Sci 3(4):1–40

Dolev S, Welch JL (1997) Wait-free clock synchronization. Algorithmica 18(4):486–511

Dolev S, Welch JL (2004) Self-stabilizing clock synchronization in the presence of byzantine faults. J ACM 51:780–799

Galbiati G, Maffioli F, Morzenti A (1994) A short note on the approximability of the maximum leaves spanning tree problem. Inf Process Lett 52(1):45–49

Garey MR, Johnson DS (1979) Computers and intractability, a guide to the theory of NP-completeness. Freeman, New York

Gartner FC (2003) A survey of self-stabilizing spanning-tree construction algorithms. Technical report, EPFL

Gopal AS, Perry KJ (1993) Unifying self-stabilization and fault-tolerance (preliminary version). In: Proceedings of the 12th annual ACM symposium on principles of distributed computing, pp 195–206

Johnen C, Tixeuil S (2003) Route preserving stabilization. In: Proceedings of the 6th international symposium on self-stabilizing systems. LNCS, vol 2704. Springer, Berlin, pp 184–198

Kamei S, Kakugawa H (2007) 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. IEEE Comput Soc, Los Alamitos, p 224

Lu HI, Ravi R (1998) Approximating maximum leaf spanning trees in almost linear time. J Algorithms 29:132–141

Papatriantafilou M, Tsigas P (1997) On self-stabilizing wait-free clock synchronization. Parallel Process Lett 7(3):321–328

Raei H, Tabibzadeh M, Ahmadipoor B, Saei S (2009) 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. IEEE Comput Soc, Los Alamitos, pp 526–530

Solis-Oba R (1998) 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In: Proceedings of the 6th annual European symposium on algorithms. LNCS, vol 1461. Springer, Berlin, pp 441–452

Tixeuil S (2009) Self-stabilizing algorithms. In: Algorithms and theory of computation handbook, 2nd edn. Chapman & Hall/CRC applied algorithms and data structures. Taylor & Francis, London.