Self-stabilizing minimum-degree spanning tree within one from the optimal degree

Lélia Blina, Maria Gradinariu Potop-Butucarub, Stephane Rovedakisc

aUniversité d’Evry, France
LIP6-CNRS UMR 7606, France

bINRIA REGAL, France
Univ. Pierre & Marie Curie - Paris 6,
LIP6-CNRS UMR 7606, France

cUniversité d’Evry, IBISC - EA 4526, 91000 Evry, France

Abstract

We propose a self-stabilizing algorithm for constructing a Minimum-Degree Spanning Tree (MDST) in undirected networks. Starting from an arbitrary state, our algorithm is guaranteed to converge to a legitimate state describing a spanning tree whose maximum node degree is at most $\Delta^* + 1$, where $\Delta^*$ is the minimum possible maximum degree of a spanning tree of the network.

To the best of our knowledge our algorithm is the first self-stabilizing solution for the construction of a minimum-degree spanning tree in undirected graphs. The algorithm uses only local communications (nodes interact only with the neighbors at one hop distance). Moreover, the algorithm is designed to work in any asynchronous message passing network with reliable FIFO channels. Additionally, we use a fine grained atomicity model (i.e., the send/receive atomicity). The time complexity of our solution is $O(mn^2 \log n)$ where $m$ is the number of edges and $n$ is the number of nodes. The memory complexity is $O(\delta \log n)$ in the send-receive atomicity model ($\delta$ is the maximal degree of the network).

Key words: Self-stabilization, minimum degree spanning tree, message passing networks.

1. Introduction

The spanning tree construction is a fundamental problem in the field of distributed network algorithms being the building block for a broad class of fundamental services: communication primitives (e.g., broadcast or multicast), deadlock resolution or mutual exclusion. The new emergent distributed systems such as ad-hoc networks, sensor or peer-to-peer networks zoom on the cost efficiency of distributed communication structures and in particular spanning trees. The main issue in ad-hoc network systems for example is the communication cost. If the communication overlay contains nodes with large degree then undesirable effects can be observed: congestion, collisions or traffic burst. Moreover, these nodes are also the first targets in security attacks. In such case, the construction of reliable spanning trees in which the degree of a node is the lowest possible is needed.

Interestingly, in peer-to-peer networks the motivation for the construction of minimum degree trees is motivated by the nodes (users) welfare. Each communication on behalf of other nodes in the network minimizes the possibility of a node to use the bandwidth for its own interests. As immediate consequence: nodes are incited to cheat on their real bandwidth or to have a free ridding behavior and illegally exploit the local traffic.

This work was partially founded by ANR projects SHAMAN and SPADES.

Email addresses: lelia.blin@lip6.fr (Lélia Blin), maria.gradinariu@lip6.fr (Maria Gradinariu Potop-Butucaru), stephane.rovedakis@ibisc.univ-evry.fr (Stephane Rovedakis)

An extended abstract of this work has been published in [1].

Preprint submitted to Elsevier

November 30, 2012
Self-stabilization as sub-field of fault-tolerance was first introduced in distributed computing area by Dijkstra in 1974 [2, 3]. A distributed algorithm is self-stabilizing if, starting from an arbitrary state, it guarantees to converge to a legitimate state in finite number of steps and to remain in a legitimate set of states thereafter. The property of self-stabilization enables a distributed algorithm to recover from a transient fault regardless of its initial state.

A broad class of self-stabilizing spanning trees have been proposed so far: by example [4, 5, 6] proposed BFS trees, [7, 8] compute minimum diameter spanning trees, and [9] calculates minimum weight spanning trees. A detailed survey on the different self-stabilizing spanning tree schemes can be found in [10]. Recently, [11] propose solutions for the construction of constant degree spanning trees in dynamic settings. To our knowledge there is no self-stabilizing algorithm for constructing minimum degree spanning trees.

This paper tackles the self-stabilizing construction of minimum-degree spanning tree in undirected graphs. More precisely, let $G = (V, E)$ be a graph. Our objective is to compute a spanning tree of $G$ that has minimum degree among all spanning trees of $G$, where the degree of the tree is the maximum degree of its nodes. In fact, since this problem is NP-hard (by reduction from the Hamiltonian path problem), we are interested in constructing a spanning tree whose degree is within one from the optimal. This bound is achievable in polynomial-time as proved by Fürrer and Raghavachari [12, 13] who describe an incremental sequential algorithm that constructs a spanning tree of degree at most $\Delta^* + 1$, where $\Delta^*$ is the degree of a MDST. Blin and Butelle proposed in [14] a distributed version of the algorithm given by Fürrer and Raghavachari. The algorithm in [14] uses techniques similar to those in [15] for controlling and updating fragment sub-trees of the currently constructed spanning trees. None of the previously mentioned solutions are self-stabilizing.

Our results. We describe a self-stabilizing algorithm for constructing a minimum degree spanning tree in arbitrary networks. Our contribution is twofold. First, to the best of our knowledge our algorithm is the first self-stabilizing approximate solution for the construction of a minimum-degree spanning tree in undirected graphs. The algorithm converges to a legitimate state describing a spanning tree whose maximum node degree is at most $\Delta^* + 1$, where $\Delta^*$ is the minimum possible degree of a spanning tree of the network. Note that computing $\Delta^*$ is NP-hard. The algorithm uses only local communications — nodes interact only with their one hop neighbors — which makes it suitable for large scale and scalable systems. Our algorithm is designed to work in any asynchronous message passing network with reliable FIFO channels. Additionally, we use a fine grained atomicity model (i.e., send/receive atomicity) defined for the first time in [7]\textsuperscript{2}.

Secondly, our approach is based on the detection of fundamental cycles (i.e., cycles obtained by adding one edge to a tree) contrary to the technique used in [14] that perpetually updates membership information with respect to the different fragments (sub-trees) co-existing in the network. As a consequence, and in contrast with [14], our algorithm is able to decrease simultaneously the degree of each node of maximum degree. The time complexity of our solution is $O(mn^2 \log n)$ where $m$ is the number of edges of the network and $n$ the number of nodes. The memory complexity is $O(\log n)$ in a classical message passing model and $O(\delta \log n)$ in the send/receive atomicity model ($\delta$ is the maximal degree of the network). The maximal length of messages used by our algorithm is $O(n \log n)$ bits. Table 1 compares our algorithm with the existing (non stabilizing) solutions.

Paper road-map. The paper is organized as follows. The next section introduces the notations and motivates the information needed to solve the problem. Section 3 describes the model considered whereas Section 4 presents the detailed description of our algorithm, then Section 5 and 6 contain the performance and complexity algorithm analysis respectively. The last section of the paper resumes the main results and outlines some open problems.

\textsuperscript{2}The send/receive atomicity is the message passing counter-part of the read/write atomicity defined for the register model [3].
Definition 1 (Improvement and Improving edge [13]). Let $e = \{u, v\}$ be an edge of $G$ that is not in $T$. Let $C_e$ be the unique cycle generated, called fundamental cycle, when $e$ is added to $T$. Suppose there is a node $w$ of degree $k$ in $C_e$ while the degrees of nodes $u$ and $v$ are at most $k - 2$. An improvement to $T$ is the modification of $T$ by adding the edge $e$, called improving edge, to $T$ and deleting one of the edges in $C_e$ incident to $w$.

Definition 2 (Blocking node [13]). Let $T$ be a spanning tree of degree $k$. Let $e = \{u, v\} \not\in T$ be an edge of $G$. Suppose $w$ is a node of degree $k$ in the cycle $C_e$ generated by adding $e$ to $T$. If $\deg_T(u) \geq k - 1$ then $u$ is a blocking node for $C_e$.

2.2. A sequential algorithm with the best approximation

The Minimum Degree Spanning Tree problem is NP-Hard to solve optimally. Fürer and Raghavachari proposed in [12, 13] an incremental sequential approximation algorithm that constructs in polynomial time a spanning tree of degree at most $\Delta^* + 1$, where $\Delta^*$ is the degree of a MDST.

The main idea of their algorithm is as follows: (i) start from an arbitrary spanning tree, (ii) compute the maximum degree of the spanning tree, (iii) if there is an improving edge $e$ then swap $e$ with an edge in $C_e$ incident to a maximum degree node and go back to step (ii). Otherwise, in order to obtain an improving edge try to make non blocking a blocking node in a fundamental cycle which contains a maximum degree node (this may lead to a sequence of blocking node degree reductions). If an improving edge is obtained then go back to step (iii), otherwise there is no possible improvement and return the current tree.

2.3. Information needed in decentralized environment

The sequential algorithm proposed in [12, 13] need the computation of the maximum degree node in the current spanning tree. The computation of this information in a decentralized way require a diffusion in the network. In the following we show that any algorithm based on edge swaps in fundamental cycles that does not use the maximum degree node or some additional information may return a maximum degree greater than $\Delta^* + 1$.

We give a counter-example which shows that all swaps are not equivalent. Consider the example shown in Figure 1(a). Assume nodes $u_i$ and $v_i$ blocking nodes for the fundamental cycle of $\{u_i, v_i\}$. In order to
reduce the degree of $x$, at least one pair of nodes $u_i, v_i$ should be promoted to a non blocking state. Edges $b_i$ and $c_i$ are respectively improving edges for $C_{b_i}$ and $C_{c_i}$. Assume edges $c_i$ are used to perform improvements. This makes only $u_i$ non blocking and no more improvement is possible because edges $b_i$ and $a_i$ contain a blocking node, as shown in Figure 1(b). On the other hand if $c_i$ are replaced by $b_i$, this leads to $u_i$ and $v_i$ non blocking and then an additional improvement reduces the degree of $x$ as shown in Figure 1(c). The degree of $x$ will be one from optimal as shown in Figure 1(d).

3. Model

We borrow the model proposed in [7]. We consider an undirected connected network $G = (V,E)$ where $V$ is the set of nodes and $E$ is the set of edges. Nodes represent processors and edges represent communication links. Each node in the network has an unique identifier. For a node $v \in V$, we denote the set of its neighbors $\mathcal{N}(v) = \{u : \{u,v\} \in E\}$ ($\{u,v\}$ denotes the edge between the node $u$ and its neighbor $v$). The size of the set $\mathcal{N}(v)$ is the degree of $v$.

We consider a static topology. The communication model is asynchronous message passing with FIFO channels (on each link messages are delivered in the same order as they have been sent)\(^3\). We use a refinement of this model: the send/receive atomicity ([7]). In this model each node maintains a local copy of the variables of its neighbors, refreshed via special messages (denoted in the sequel InfoMsg) exchanged periodically by neighboring nodes. We assume reliable channels, that is there are no corrupted or lost messages. A local state of a node is the value of the local variables of the node, the copy of the local variables of its neighbors and the state of its program counter. A configuration of the system is the cross product of the local states of all nodes in the system plus the content of the communication links. An atomic step at node $p$ is an internal computation based on the current value of $p$’s local state and a single communication operation (send/receive) at $p$. A computation of the system is an infinite sequence of configurations, $e = (c_0, c_1, \ldots, c_i, \ldots)$, where each configuration $c_{i+1}$ follows from $c_i$ by the execution of at least a single atomic step by at least one node.

\(^3\)We use FIFO channels in order to ensure the correctness and the polynomial complexity of our solution.
Faults and self-stabilization. In the sequel we consider the system can start in any configuration. That is, the local state of a node can be corrupted. Note that we don’t make any assumption on the bound of corrupted nodes. In the worst case all the nodes in the system may start in a corrupted configuration. In order to tackle these faults we use self-stabilization techniques.

Definition 3 (self-stabilization). Let $\mathcal{L}_A$ be a non-empty legitimacy predicate\(^4\) of an algorithm $A$ with respect to a specification predicate $\text{Spec}$ such that every configuration satisfying $\mathcal{L}_A$ satisfies $\text{Spec}$. Algorithm $A$ is self-stabilizing with respect to $\text{Spec}$ iff the following two conditions hold:

1. Every computation of $A$ starting from a configuration satisfying $\mathcal{L}_A$ preserves $\mathcal{L}_A$ (closure).
2. Every computation of $A$ starting from an arbitrary configuration contains a configuration that satisfies $\mathcal{L}_A$ (convergence).

Minimum Degree Spanning Tree (MDST). A legitimate configuration for the MDST is a configuration that outputs an unique spanning tree of minimum degree. Since computing a $\Delta^*$ minimum degree spanning tree in a given network is NP-hard we propose in the following a self-stabilizing MDST with maximum node degree at most $\Delta^* + 1$.

4. A self-stabilizing MDST Algorithm

The main ingredients used by our MDST approximation algorithm are: (1) a module maintaining a spanning tree; (2) a module for computing the maximum node-degree of the spanning tree; (3) a module for computing the fundamental cycles and (4) a procedure for reducing the maximum node-degree of the spanning tree based on the fundamental cycles computed by the previous module.

One challenge is to design and run these modules concurrently, in a self-stabilizing manner. Note that the core of our algorithm is the reduction procedure that aims at repetitively reducing the degree of the spanning tree until getting a spanning tree of degree at most $\Delta^* + 1$.

The following section provides the detailed description of our algorithm. The formal description of the algorithm\(^5\) can be found in Figure 3, with its sub-procedures in Figures 6 and 2.

4.1. Data Structures and Elementary procedures

Variables. This short section lists all the variables used by the algorithm. For any node $v \in V(G)$, $N(v)$ (we assume an underlying self-stabilizing protocol that regularly updates the neighbors set) denotes the set of all neighbors of $v$ in the network $G$, $\text{ID}_v \in \mathbb{N}$ is the unique identifier of $v$ and $\text{deg}_v$ is the degree of $v$ in the tree (i.e., $\text{deg}_v = \{|u : u \in N(v) \land \text{is tree edge}(v, u)|\}$). Nodes repeatedly send their variables to each of their neighbors $u$ via an $\text{InfoMsg}$ and update their local variables upon reception of this type of message from a neighbor. Each node $v$ maintains the following variables:

- **Integer type variables:**
  - root$_v$: the ID of the root of the spanning tree (computed by node $v$);
  - parent$_v$: the parent ID of $v$ in the spanning tree;
  - distance$_v$: the distance of $v$ to the root of the tree;
  - $\text{dmax}_v$: local estimation of $k$ (maximum degree of the spanning tree), it is updated upon the reception of an $\text{InfoMsg}$ message. A change on $k$ is detected via the color$_{\text{tree}}_v$ variable. This information is further disseminated in the network via $\text{InfoMsg}$ messages;
  - $\text{dmax}_{\text{t}v}$: local estimation of the maximum degree in the subtree rooted at $v$, it is updated upon the reception of an $\text{InfoMsg}$ message;

\(^4\)A legitimacy predicate is defined over the configurations of a system and is an indicator of its correct behavior.

\(^5\)In the proposed algorithms operator $\oplus$ concatenates lists.
• **Boolean type variables:**
  - `edge_status_v[u]`: is true when `{u, v}` is an edge of the tree;
  - `color_tree_v`: used to track any change on `dmax_v`;

**Messages.** The messages used by our algorithm are the following:

- `(InfoMsg, root_v, parent_v, distance_v, dmax_v, dmax_t_v, deg_v, edge_status_v[u], color_tree_v)`: this message is used to gossip the local variables of a node.
- `(Search, init_edge, idblock, path)`: it is used to find the fundamental cycle induced by a non tree edge, where `init_edge` is the initiate non tree edge, `idblock` is the ID of a blocking node and `path` is the fundamental cycle.
- `(Remove, init_edge, deg_max, target, path)`: it is used to reduce the degree of a maximum degree node by deleting an adjacent edge. This message is also used to reverse the edge orientation in the fundamental cycle. Information transported are: `init_edge` is the initiate non tree edge, `target` is the edge to be deleted, `deg_max` is the maximum degree of `target` extremities and `path` is the fundamental cycle.
- `(Back, init_edge, target, path)`: change edge orientation in the fundamental cycle after an improvement, where `init_edge` is the initiate non tree edge, `target` is the deleted edge and `path` is the fundamental cycle.
- `(Deblock, idblock)`: it is used to change the state of a blocking node, where `idblock` is the ID of a blocking node.
- `(Reverse, target, path)`: it is used to erase the modifications done in a fundamental cycle because of concurrent improvements, where `target` is the deleted edge.
- `(UpdateDist, dist)`: update the distance of fundamental cycle nodes, where `dist` is the distance from the tree root of the sender.

Note that the `path` information is never stored at a node, therefore it is not listed as local variable of a particular node. However this information is carried by different messages. Therefore the complexity of our solution in the length of the network buffers is at least $O(n \log n)$ (the maximal length of the `path` chain) as explained in the complexity section.

**4.2. Elementary building blocks**

In this section, we provide first a detailed description of the underlying modules for our degree reduction algorithm, namely: a spanning tree module, a module that computes and disseminates the maximal degree of the spanning tree and finally a module that detects fundamental cycles. We conclude the section by the detailed presentation of the degree reduction module.

**4.2.1. Spanning tree module**

The algorithm described below is a simplification of the BFS algorithm proposed in [4]. Each node `v` maintains three variables: the local known ID of the tree root, a pointer to `v`’s parent and the distance of `v` to the spanning tree root. The output tree is rooted at the node having the minimum root value.

The algorithm uses two rules formally specified below. The first rule, “correction parent”, enables the update of the locally known root: for a node `v` if a neighbor `u` has a lower root ID than `v` then `v` changes its root to `u`’s root, and `u` becomes the parent of `v`. If several neighbors verify this property then `v` will choose as parent the node with the minimal ID. This choice is realized by the argmin operation. The second rule, “correction root”, creates a root if the neighborhood of a node has an incoherent state. In a coherent state, the distance in the BFS tree with respect to the root should be the distance of the parent plus 1. In the case of the root this distance should be 0. Additionally, the parent of a node should be a neighbor.

We give below the predicates, procedures and rules used for the spanning tree construction.
Predicates. The predicates in the spanning tree module at node v are the following:

- better_parent(v) ≡ \( \bigvee_{u \in N(v)} \) root_{v} > root_{u}
- coherent_parent(v) ≡ (parent_v \in N(v) \cup \{v\}) \land (root_v = \text{root}_\text{parent}_v)
- coherent_distance(v) ≡ (parent_v \neq \text{ID}_v \land \text{distance}_v = \text{distance}_\text{parent}_v + 1) \lor (parent_v = \text{ID}_v \land \text{distance}_v = 0)
- new_root_candidate(v) ≡ \neg \text{coherent_parent}(v) \lor \neg \text{coherent_distance}(v)
- is_tree_edge(v, u) ≡ parent_v = \text{ID}_u \lor parent_u = \text{ID}_v

Procedures. The procedures in the spanning tree module at node v are the following:

- change_parent_to(v, u) ≡ root_v := \text{root}_u; parent_v := \text{ID}_u; \text{distance}_v := \text{distance}_u + 1
- create_new_root(v) ≡ root_v := \text{ID}_v; parent_v := \text{ID}_v; distance_v := 0

Rules. The rules in the spanning tree module at node v are the following:

- **R\text{Correct}_p:** (correction parent)
  
  If \( \neg \text{new_root_candidate}(v) \land \text{better_parent}(v) \) then change_parent_to(v, argmin\{root_{u} : u \in N(v)\});

- **R\text{Correct}_r:** (correction root)
  
  If \( \text{new_root_candidate}(v) \) then create_new_root(v);

4.2.2. Maximum degree module

Our construction of minimum-degree spanning tree from an arbitrary spanning tree requires at every node v the knowledge of: (1) which of its adjacent edges belongs to the spanning tree; (2) the maximum degree of the current spanning tree.

Note that both information can be computed using only local communication as it will be described in the sequel. The coherent maintenance of the edges part of the spanning tree is realized via the variable edge_status_{v} [u], updated by InfoMag messages. This process is similar with the maintenance of dynamic trees [16].

Our protocol uses a Propagation of Information with Feedback (PIF) strategy (see [17, 18] for more details). In order to propose a self-contained solution, we give in the following a way to compute and to disseminate the maximum node-degree of the current spanning tree. The proposed rules use two variables at a node v: dmax_{tv} and dmax_{v}. The first one is used to inform the parent of v of the maximum node-degree in the subtree rooted at v. The last one is used to disseminate in the tree the correct maximum degree of the current spanning tree. Moreover, the rules described below use Predicate local_max_degree(v). This predicate returns the degree in the tree for a leaf node or the maximum value between the degree in the tree of v and the maximum degree computed in the subtree of v (maximum value between the dmax_{tv} variables of children u of v). To compute the maximum degree of the current spanning tree, a node v must stabilize first its variable dmax_{tv}. If dmax_{tv} has a value different from the value given by Predicate local_max_degree(v) then v uses Rule RMax_dFeedB to correct the value of variable dmax_{tv}. When the variable dmax_{tv} is stabilized at node v, v can execute Rules RMax_dRoot and RMax_dPro to disseminate the maximum degree of the tree by updating the value of dmax variables. Rule RMax_dRoot is used by the root of the tree to update its dmax variable and to inform its children of the maximum degree of the tree. This rule is only used in the case where dmax_v \neq dmax_{tv}. Any other node in the tree uses Rule RMax_dPro to update its variable dmax and to inform its children of the maximum degree of the tree. This rule is only used in the case where v has a different estimation of the maximum degree with its parent, i.e., we have dmax_v \neq dmax_{\text{parent}_v}.
Predicates. The predicates in the maximum degree module at node \( v \) are the following:

- \( \text{tree} \_\text{stabilized}(v) \equiv \neg \text{better} \_\text{parent}(v) \land \neg \text{new} \_\text{root} \_\text{candidate}(v) \)
- \( \text{local} \_\text{max} \_\text{degree}(v) \equiv \max(\text{deg}_v, \max\{\text{dmax} \_\text{t}_u : u \in N(v) \land \text{parent} \_u = v\}) \)
- \( \text{degree} \_\text{stabilized}(v) \equiv \bigwedge_{u \in N(v)} \text{dmax}_v = \text{dmax}_u \)
- \( \text{color} \_\text{stabilized}(v) \equiv \bigwedge_{u \in N(v)} \text{color} \_\text{tree}_v = \text{color} \_\text{tree}_u \)
- \( \text{locally} \_\text{stabilized}(v) \equiv \text{tree} \_\text{stabilized}(v) \land \text{color} \_\text{stabilized}(v) \)

Rules. The rules in the maximum degree module at node \( v \) are the following:

- **RMax\_dFeedB:** (max degree feedback)
  - If \( \text{tree} \_\text{stabilized}(v) \land \text{dmax} \_\text{t}_v \neq \text{local} \_\text{max} \_\text{degree}(v) \) then \( \text{dmax} \_\text{t}_v \leftarrow \text{local} \_\text{max} \_\text{degree}(v) \);

- **RMax\_dRoot:** (max degree of root)
  - If \( \text{tree} \_\text{stabilized}(v) \land \text{parent} \_v = \text{root} \_v \land \text{dmax} \_\text{t}_v = \text{local} \_\text{max} \_\text{degree}(v) \land \text{dmax}_v \neq \text{dmax} \_\text{t}_v \) then \( \text{dmax}_v \leftarrow \text{dmax} \_\text{t}_v \);

- **RMax\_dPro:** (max degree propagation)
  - If \( \text{tree} \_\text{stabilized}(v) \land \text{parent} \_v \neq \text{root} \_v \land \text{dmax} \_\text{t}_v = \text{local} \_\text{max} \_\text{degree}(v) \land \text{dmax}_v \neq \text{dmax} \_\text{parent} \_v \) then \( \text{dmax}_v \leftarrow \text{dmax} \_\text{parent} \_v \);

Note that the maximal degree of the tree changes during the execution of the algorithm via the degree reduction procedure. When a node of maximum degree has its degree decreased by one, the algorithm uses \( \text{color} \_\text{tree}_v \) variable to detect this change (see lines 37 and 43, Figure 6). Additionally, a node that detects an incoherence in its neighborhood related to the maximum degree blocks the construction of the minimum-degree spanning tree until the neighborhood becomes \( \text{locally} \_\text{stabilized} \) (all neighbors have the same value for their \( \text{color} \_\text{tree} \) variable).

### 4.2.3. Fundamental cycle detection module

We recall that given \( G \) a network, \( T \) a spanning tree of \( G \) and \( e = \{u, v\} \) an edge of \( G \) that is not in \( T \), the cycle \( C_e \) generated by adding \( e \) to \( T \) is called fundamental.

Computing a fundamental cycle at an arbitrary node \( v \) is performed by Procedure **Cycle\_Search** (see Figure 2). For each non tree edge \( \{u, v\} \) such that \( \text{ID}_v < \text{ID}_u \) (\( v \) is the initiator for the non tree edge \( \{u, v\} \)), node \( v \) initiates a DFS in the current spanning tree \( T \) to discover the path in the tree between \( u \) and \( v \) (Figure 2, lines 1-3). This DFS uses messages of type **Search**. The DFS propagation (Figure 2, lines 4-6) discovers only the tree edges in the fundamental cycle of \( \{u, v\} \). It updates \( \text{path} \) by appending the discovered node and its degree in the tree.

The DFS search propagation stops (Figure 2, lines 7-8) when the other extremity of the edge (i.e., \( u \)) is reached and \( \text{path} \) contains only the tree edges in the fundamental cycle of \( \{u, v\} \).

**Cycle\_Search**(id)
1   if locally\_stabilized(v) then
2       \forall u \in N(v) with \text{edge}\_\text{status}_u[u] = \text{false} \land \text{ID}_v < \text{ID}_u: \text{initiate a DFS propagation of a Search message crossing}
3       only tree edges by sending \langle \text{Search}, (u, v), id, \{v, \text{deg}_v\}\rangle \text{ to } w \in N(v) \text{ s.t. \text{edge}\_\text{status}_w[w] = true}
4   Upon receipt \langle \text{Search}, (x, y), id, \text{path} \rangle \land \text{locally}\_\text{stabilized}(v) \text{ from } u:
5       if \( v \neq x \) then
6           DFS propagation on tree edges of \langle \text{Search}, (x, y), id, \text{path} \oplus \{v, \text{deg}_v\}\rangle
7       else
8           if \neg \text{edge}\_\text{status}_u[y] then
9               Action\_on\_Cycle(id, y, \text{path}, u);

Figure 2: Fundamental cycle detection at node \( v \)
4.2.4. Degree reduction module

For all edges that do not belong to the current tree $T$ our algorithm proceeds as follows. For each of these edges $e$, one of its adjacent nodes computes the fundamental cycle $C_e$, and checks whether $e$ is an improving edge, and whether the maximum degree in $C_e$ is equal to $\deg(T)$. If the two latter conditions are satisfied, then $e$ is swapped with one edge of $C_e$ incident to a node of maximum degree in $T$. This operation is repeated until there is no more improving edge $e$ with degree of $C_e$ equal to the degree of the current tree. If a blocking node for $C_e$ is encountered, then the algorithm tries to reduce the degree of the blocking node, recursively, in a way similar to the one used for decreasing the degree of the nodes of maximum degree. When no more improvement is possible then the algorithm stops. Note that [13] proved that the degree of a tree for which no improvement is possible is at most $\Delta^* + 1$, where $\Delta^*$ is the degree of a MDST of $G$. In the following we describe in detail the above reduction process.
**Cycle_Search:** Repeatedly each non tree edge looks for its fundamental cycle by using the procedure `Cycle_Search` described in Section 4.2. When a non tree edge $e = \{u, v\}$ discovers its fundamental cycle (Figure 2, line 8) it checks whether it is an improving edge in procedure `Action_on_Cycle`.

**Action_on_Cycle:** If the fundamental cycle contains no maximum degree node, no improvement is possible for this fundamental cycle. Otherwise (line 9) if the fundamental cycle also contains no blocking node (line 7, i.e., $idblock = \text{NIL}$) an improvement is possible (procedure `Improve` is called, lines 12-15). If there are blocking nodes the procedure `Deblock` is started (lines 10-11). The procedure tries to improve the blocking node (lines 19-22), either via the procedure `Improve` or via the procedure `Deblock` (lines 16-18).

**Improve:** The procedure `Improve` sends a `Remove` message (lines 28-29), which is propagated along the fundamental cycle (Figure 3, lines 8-12). When the `Remove` message reaches its destination (edge $e' = \{w, z\}$), the degree and the status of $e'$ are checked. If the maximum degree or the edge status have changed, then the `Remove` message is discarded. Otherwise, the status of edge $e'$ is modified (it becomes a non-tree edge), and a message is sent along the cycle to correct the parent orientation (procedure `Reverse_Orientation` explained below). When `Remove` reaches the edge $e$ that initiated the process then $e$ is added to the tree (Figure 3, line 7). In the case when `Remove` message meets a deleted edge on its path, it carries on as if the deleted edge would be still alive.

**Reverse_Orientation:** After the remove of an edge $e'$, the orientation of the fundamental cycle must be corrected (see Figure 5). This is achieved via the circulation of two messages: `Back` and `Remove`. The procedure `Reverse_Orientation` deletes $e'$ and checks following the orientation of $e'$ whether a `Back` or a `Remove` message must be used to correct the orientation. If $e'$ is oriented opposite to the direction followed by `Remove` message, then a `Remove` message is used (Figure 5(a)) otherwise a `Back` message is used (Figure 5(b)). The result of `Reverse_Orientation` on the path from $e'$ to $e$ on the fundamental cycle of $e$ can be seen in Figure 5(c).

Note that a `Remove` (or `Back`) message propagated along the fundamental cycle may meet an edge deleted by another improvement. This implies that the fundamental cycle of $e$ has changed. In this case, a `Reverse` message is sent back to erase the modifications done in the fundamental cycle, which ends by adding the removed edge $e'$. If the changes made by the improvement are not erased this may conduct to partition the tree. Remark: if an edge deleted by another improvement is encountered in the fundamental cycle before the deletion of an edge $e'$, then the `Remove` message is discarded because the fundamental cycle has changed and the improvement must not performed (Figure 3, line 12).

**Deblock:** To perform an edge exchange with a blocking node $w \in \{u, v\}$, our algorithm starts by reducing by one the degree of $w$. For this purpose, $w$ broadcasts a `Deblock` message containing its identifier.
in all its sub-tree (see procedure Deblock). When a descendant $w'$ of $w$, adjacent to a non tree edge $f$, receives a Deblock message, it proceeds with the discovery of the fundamental cycle $C_f$ of $f$ as described in Section 4.2. If this fundamental cycle $C_f$ enables to decrease the degree of the blocking node $w$, then $w$ is not anymore blocking, and the swap $e$ with $e'$ can be performed. However, if the fundamental cycle $C_f$ does not enable to decrease the degree of the blocking node $w$, then the procedure carries on recursively by broadcasting a Deblock messages for $w'$.

![Figure 5: Illustration of Reverse_Orientation procedure: (a) when ID$_x$ > ID$_y$, (b) when ID$_x$ < ID$_y$, (c) the resulted network given by (a) and (b).](image)

Note that in order to maintain the tree stabilized we must update the distance of nodes on the path reversed after the remove of $e'$, as their distances to the tree root has changed. Therefore an UpdateDist message is diffused for all children of nodes on the reversed path.

5. Algorithm correctness

In this section prove the correctness of our protocol. The proof is divided in three parts: first we show that our spanning tree module eventually returns a spanning tree of the network, second we show that every node in the network eventually knows the maximum degree of the current spanning tree using our maximum degree module. Finally, we show that our degree reduction module performs correctly improvements in the tree and reaches the desired approximation on the maximum degree of the tree compared with the optimal solution.

We start below by defining a legitimate configuration of the system.

**Definition 4 (Legitimate state of a MDST).** A configuration of our algorithm is legitimate iff each process $v \in V$ satisfies the following conditions:

1. a spanning tree $T$ is constructed;
2. the maximum degree of $T$ is at most $\Delta^* + 1$, with $\Delta^*$ the maximum degree of the optimal solution.

In the sequel we address different aspects related to the corruption of the local state of a node: InfoMsg messages ensure the coherence between the copy of a node variables and their original values. Therefore, even if a node maintains corrupted copies their values will be corrected as soon as the node receives InfoMsg messages from each of its neighbors.

The reduction procedure and the search of fundamental cycles are frozen at a node $v$ until the neighborhood of $v$ is locally stabilized. That is, modules executed at an arbitrary node are locally ordered on priority bases: the module with maximal priority being the spanning tree construction, followed by the maximum degree and degree reduction module.

In the following we prove that the execution of the reduction procedure is blocked for a finite time (i.e., the time needed to the algorithm to compute a spanning tree). That is, we prove that the algorithm computes in a self-stabilizing manner a spanning tree in a finite number of steps.

**Lemma 1.** Starting from an arbitrary configuration, eventually all nodes share the same root value.
**Proof.** Nodes periodically exchange InfoMsg messages with their neighborhood. Therefore, even if the local copy of the neighbors variables are not identical with the original ones, upon the reception of an InfoMsg message a node corrects its local copies (via the procedure $\text{Update\_State}$). Assume w.l.o.g. two different root values coexist in the network. Let $v_1 < v_2$ be these values. Let $l$ be the number of nodes having their root variable set to $v_2$. Since the network is connected there are two neighboring nodes $p_1$ and $p_2$ such that the local root of $p_1$ is $v_1$ and the local root of $p_2$ is $v_2$. After the reception of an InfoMsg from $p_1$, $p_2$ detects its inconsistency and changes the local root value to $v_1$ and the parent variable to $p_1$ by executing the rule “correction parent”. Hence the number of root variables set to $v_2$ decreases by 1. Recursively, the number of root variables set to $v_2$ falls to 0.

Note that the root values are not modified by the other procedures of the algorithm. Eventually, all nodes in the network will have the same root value.  

**Lemma 2.** Starting from an arbitrary configuration, eventually an unique spanning tree is alive in the network.
Proof. Assume the contrary: two or more spanning trees are created. This leads to either the existence of two or more different root values or the presence of at least one cycle. Since the values of the root variable are totally ordered then by Lemma 1 one of those values will eventually win the contest which invalidates the multiple roots assumption.

Assume there is a cycle in the network. Note that there is an unique root and each node has a distance to the root which is the distance of the parent plus one. In a cycle there is at least one node such that the coherent_distance predicate returns false. This node will eventually execute the “correction root” rule. According to the proof of Lemma 1 the new root will run a competition with the existing root and the root with the lowest ID remains root while the other one modifies its root and distance variables.

Assuming that a spanning tree of the network is constructed, now we prove that the maximum degree module computes the maximum node-degree of the constructed spanning tree. Note that if a spanning tree is not constructed then there is a node v which cannot execute the maximum degree module because Predicate tree_stabilized(v) is not satisfied.

**Lemma 3.** Starting from an arbitrary configuration which contains a spanning tree $T$, eventually for every node $v$ variable $d_{\text{max}}_T(v)$ stores the maximum node-degree of the subtree rooted at $v$.

**Proof.** We assume that a spanning tree $T$ is constructed, i.e., Predicate tree_stabilized($v$) is satisfied for every node $v$. We prove the lemma by induction on the height of $T$. For any node $v$, Predicate $\text{local\_max\_degree}(v)$ returns the maximum value between the degree of $v$ in $T$ and the maximum value of $d_{\text{max}}_T$ variables of $v$'s children.

Consider the leaves $v$ of $T$. If $v$ does not satisfy the lemma then this implies that $d_{\text{max}}_T(v) \neq 1$ because $v$ is its own subtree. Since $v$ is a leaf, $v$ has no child and Predicate $\text{local\_max\_degree}(v)$ returns the degree of $v$ which is equal to one. Thus, the guard of Rule $R\text{Max}_d\text{FeedB}$ is satisfied because we have $d_{\text{max}}_T(v) \neq 1$ and $v$ can execute Rule $R\text{Max}_d\text{FeedB}$ to correct the value of $d_{\text{max}}_T(v)$ which verifies the lemma. Consider any internal node $v$ of $T$. We assume by induction hypothesis that every node in the subtree rooted at $v$ has a correct value stored in $d_{\text{max}}_T$ variable. Assume $v$ does not satisfy the lemma, this implies that $d_{\text{max}}_T(v) \neq \text{local\_max\_degree}(v)$. $v$ can execute Rule $R\text{Max}_d\text{FeedB}$ to correct its variable $d_{\text{max}}_T$ and the lemma is verified. Therefore, after a finite number of steps for every node $v$ in $T$ the variable $d_{\text{max}}_T(v)$ stores the maximum node-degree of the subtree rooted at $v$.

**Lemma 4.** Starting from an arbitrary configuration which contains a spanning tree $T$, eventually for every node $v$ variable $d_{\text{max}}_v$ stores the maximum node-degree of $T$.

**Proof.** We assume that a spanning tree $T$ is constructed, i.e., Predicate tree_stabilized($v$) is satisfied for every node $v$. We prove the lemma by induction on the height of $T$. Moreover, according to Lemma 3 we assume that for every node $v$ in $T$ variable $d_{\text{max}}_T$ stores the maximum node-degree of the subtree rooted at $v$, i.e., we have $d_{\text{max}}_T(v) = \text{local\_max\_degree}(v)$.

Consider the root $v$ of $T$ at height 0. Since $v$ is the root of $T$, variable $d_{\text{max}}_T(v)$ contains the maximum node-degree rooted at $v$ that is the maximum node-degree of $T$. If $v$ does not satisfy the lemma then we have $d_{\text{max}}_v \neq d_{\text{max}}_T(v)$. Thus, $v$ can execute Rule $R\text{Max}_d\text{Root}$ to correct its $d_{\text{max}}_v$ variable and to verify the lemma. Assume by the induction hypothesis that every node $v$ at height $i$ has the maximum node-degree stored in variable $d_{\text{max}}_v$. Consider any node $v$ at height $i + 1$. If $v$ does not satisfy the lemma then this implies that $v$ has a value stored in variable $d_{\text{max}}_v$ different from its parent which has a correct value by induction hypothesis, i.e., we have $d_{\text{max}}_v \neq d_{\text{max}}_{\text{parent\_v}}$. Thus, $v$ can execute Rule $R\text{Max}_d\text{Pro}$ to correct its $d_{\text{max}}_v$ variable and to verify the lemma. Therefore, after a finite number of steps for every node $v$ in $T$ the variable $d_{\text{max}}_v$ stores the maximum node-degree of the spanning tree $T$.

Now we prove that our algorithm maintains the same performance as the algorithm in [13] and converges to a legitimate configuration verifying the hypothesis of Theorem 1 ([13]).
Theorem 1. (Fürer and Raghavachari [13]) Let $T$ be a spanning tree of degree $\Delta$ of a graph $G$. Let $\Delta^*$ be the degree of a minimum-degree spanning tree. Let $S$ be the set of vertexes of degree $\Delta$ in $T$. Let $B$ be an arbitrary subset of vertexes of degree $\Delta - 1$ in $T$. Let $S \cup B$ be removed from the graph, breaking the tree $T$ into a forest $F$. Suppose $G$ satisfies the condition that there are no edges between different trees in $F$. Then $\Delta \leq \Delta^* + 1$.

We call improvement a step of our algorithm in which the degree of at least one node of maximum degree is decreased by 1, without increasing the degree of a node which already has the maximum degree, nor the degree of a blocking node.

Lemma 5. When applying procedure Improve, our algorithm makes an improvement.

Proof. Let $e = \{u, v\}$ be an improving edge for the current tree $T$ of maximum degree $k$. Before adding $e$ to $T$ in the procedure Improve, some extremity of the edge $e$ sends a Remove message along $C_e$ to delete some edge $e' = \{x, y\}$ of the cycle, adjacent to a node of degree $k$ or $k - 1$. This message Remove is routed along $C_e$. When it reaches one extremity of edge $e'$, here are two cases:

Case 1: $e'$ is still a tree edge of degree $k$ or $k - 1$. Then, according to procedure Improve, edge $e'$ is removed from $T$, and $u$ or $v$ is informed by a Remove or Back message that $e$ can be added to the tree $T$. In this case, we have an improvement.

Case 2: $e'$ is no more a tree edge of degree $k$ or $k - 1$. This implies that the degree of $C_e$ has been decreased (by a concurrent improving edge), yielding an improvement. In this case, according to procedure Improve, the message Remove is discarded, for preserving the spanning tree property. In both cases, an improvement occurred. 

To obtain the same performance as algorithm in [13], it is necessary in some configuration to make a blocking node non blocking because this leads to reduce the degree of a maximum degree node. The following lemma states that if it is possible to make a blocking node non blocking, then our algorithm makes non blocking a blocking node without increasing the degree of the tree.

Lemma 6. Let $w$ be a blocking node which can be made non blocking. Then procedure Deblock$(w)$ reduces the degree of $w$ by 1, without increasing the degree of neither a node with maximum degree, nor a blocking node.

Proof. The proof proceeds by induction on the number of calls to procedure Deblock. If at the first call to Deblock there is an improving edge in the sub-tree of $w$ (according to [13] it is sufficient to look for improving edge in the sub-tree of $w$) then an improvement is performed by the procedure Improve reducing the degree of $w$ according to Lemma 5. If there is no improving edge but at least one blocking node adjacent to a non tree edge in the sub-tree of $w$ which contains $w$ in its fundamental cycle, then the procedure Deblock is recursively called by these ones according to procedure Deblock. So if an improvement is possible, this improvement is propagated to $w$ by a sequence of improvements. Note that each improvement can only increase the degree of nodes with a degree $\leq k - 2$ according to Lemma 5, so procedure Deblock$(w)$ can not increase the degree of neither a node with maximum degree, nor a blocking node.

During an improvement in a fundamental cycle, our algorithm changes a part of the spanning tree by modifying the parent of some nodes in the fundamental cycle (edge reversal process). The following lemma proves that during an improvement, our algorithm does not destabilize the spanning tree module.

Lemma 7. During the edge reversal process, Predicate tree_stabilized(v) is satisfied for each node $v$ in the tree.

Proof. We must consider every node $v$ in the tree (except the root) which takes part in the edge reversal process. The edge reversal process is achieved by the Remove, Back and Reverse messages. When one of these messages is received by a node $v$, the message is handled if and only if Predicate locally_stabilized(v) is true (i.e., if Predicates tree_stabilized(v) and color_stabilized(v) are true). We assume that a spanning
tree is constructed (see Lemma 2) and the maximum degree node of the tree is computed, this implies that every node \( v \) has a parent and a coherent distance with its parent (i.e., Predicates tree\_stabilized(\( v \)) and color\_stabilized(\( v \)) are true for every node \( v \)). To simplify the proof, we focus on Predicate tree\_stabilized(\( v \)) instead of Predicate locally\_stabilized(\( v \)) for any node \( v \). Indeed, we assume that Predicate color\_stabilized(\( v \)) is true initially this predicate remains true in a fundamental cycle during the edge reversal process because we consider FIFO channels. Thus, the reduction of the maximum degree node in the tree is taken into account at the end of the edge reversal process (see the handling of InfoMsg message). When a node \( v \) executes the edge reversal procedure then \( v \) takes a new parent and updates its distance. To prove the lemma, we must show that during the edge reversal process every node \( v \) has a parent in its neighborhood and has a distance equal to its parent distance plus one (i.e., Predicate tree\_stabilized(\( v \)) is true).

Consider any node \( v \) which satisfies Predicate tree\_stabilized(\( v \)) and executes the edge reversal process. According to the algorithm description of the edge reversal procedure, a node takes as new parent its predecessor in a fundamental cycle, thus it can only take a neighbor as new parent. Now we verify if the distance coherency is maintained. There is two cases, the edge between \( v \) and its predecessor \( p \) in the fundamental cycle is (i) a non-tree edge or (ii) a tree edge. In case (i), \( v \) sends to its parent a Reverse message to reverse the edge orientation between \( v \) and the target edge. Thus, Predicate tree\_stabilized(\( v \)) is still satisfied because \( v \) do not change its parent. The update of the rest of the path between \( v \) and the target edge is the same as case (ii) (see the handling of Reverse message and Procedure Reverse\_Aux). In case (ii), \( v \) changes its parent variable and updates its distance. Then, \( v \) sends first to its new parent \( p \) a Remove or Back message and after sends to its neighborhood an InfoMsg message (see the handling of Remove, Back and Reverse messages). Thus, \( v \) satisfies Predicate tree\_stabilized(\( v \)) after the execution of the edge reversal procedure, but we must show that \( p \) satisfies Predicate tree\_stabilized(\( p \)) too since we create a temporary cycle between \( v \) and \( p \).

Since we use a fine grained atomicity model, \( p \) maintains a copy of the old distance of its parent \( v \). Moreover, we consider FIFO channels in the network thus the Remove, Back or Reverse message is received by \( p \) before the InfoMsg message. As we consider that Predicate tree\_stabilized(\( p \)) is satisfied initially, after the modification of the parent and distance variables of \( v \) the node \( p \) still satisfies Predicate tree\_stabilized(\( p \)) because \( p \) uses the old state of \( v \). Thus, \( p \) can execute the edge reversal process as described for \( v \). When \( p \) receives the InfoMsg message from \( v \), then \( p \) updates its local copy of \( v \)'s state but as \( v \) is no more \( p \)'s parent this does not affect the satisfiability of Predicate tree\_stabilized(\( p \)). Note that to maintain coherent distances, the UpdateDist message is propagated in the subtree of a node which has changed its parent and each node receiving this message updates its distance accordingly with its parent (see the handling of the UpdateDist message).

Using the same argument, one can show by induction on the length of a fundamental cycle that Predicate tree\_stabilized(\( v \)) is satisfied by every node \( v \).

Since we perform improvements in all fundamental cycles concurrently we must ensure that at least one improvement is achieved. The following lemma states that if improvements are done in non-disjoint fundamental cycles (i.e., fundamental cycles with common paths) then at least one improvement is performed.

**Lemma 8.** Let \( C_1 \) and \( C_2 \) two non-disjoint fundamental cycles. If procedure Improve is applied in parallel in \( C_1 \) and \( C_2 \) then at least one improvement is performed.

**Proof.** We assume that a coherent spanning tree is constructed, otherwise the improvement procedures are not applied because Predicate locally\_stabilized is not satisfied (see the algorithm description). During the improvement in a fundamental cycle, the orientation of a part of the fundamental cycle is reversed using Remove or Back message. Improvements in non-disjoint fundamental cycles can interfere during the edge reversal procedure. Let \( C \) be a fundamental cycle and \( w, x \in C \) be respectively the node which starts (deletes the edge adjacent to the maximum degree node) and stops (adds the non-tree edge to the tree) the reverse orientation procedure. Let \( P(C) \) be the path reversed in the spanning tree between \( w \) and \( x \) during an improvement in the fundamental cycle \( C \). To show the lemma, we consider w.l.o.g. two non-disjoint fundamental cycles \( C_1 \) and \( C_2 \) and we prove that in the following several cases at least one improvement is
Cycle status on status e

Improve tree T possible improvement for T

Action to procedure Assume the contrary. We assume there is a tree structure T. When our algorithm reaches a legitimate configuration, there is no more possible improvement.

In the following we prove that a legitimate configuration verifies the hypothesis of Theorem 1.

Proof. Theorem 2. The algorithm returns a spanning tree of G with degree at most $\Delta^* + 1$.

Proof. In the following we prove that a legitimate configuration verifies the hypothesis of Theorem 1. That is, when our algorithm reaches a legitimate configuration, there is no more possible improvement. Assume the contrary. We assume there is a tree structure $T$ with maximum degree $k$. Suppose there is a possible improvement for $T$, by performing the swap between the edges $e_1 \notin T$ and $e_2 \in T$ to obtain the tree $T' = T \cup \{e_1\} \setminus \{e_2\}$ and assume algorithm does not perform this improvement. This implies that edge $e_1$ has a degree equal to $k$, otherwise as showed by Lemmas 5 and 6 the node of minimum id of $e_1$ will perform an improvement with procedure Improve or run DeblocK to reduce the degree of $e_1$, according to procedure Action_on_Cycle. This contradicts the fact that $e_1$ can be used to perform an improvement, and so if there is an improvement for $T$ then the algorithm will perform this improvement.

Lemma 9 (convergence). Starting from an arbitrary configuration, the algorithm returns a legitimate configuration.

Proof. Our algorithm is a parallel composition of three layers: a spanning tree construction, a maximum degree computation and a degree reduction layer. According to Lemmas 2 and 4, a spanning tree is constructed and its maximum degree is computed starting from an arbitrary configuration. Note that the maximum degree module cannot destabilize the spanning tree module because it does not modify the parent variable of a node. However, the degree reduction module modifies the tree (parent variables) and its maximum degree (dmax variables). According to Lemma 7, the degree reduction module does not destabilize the spanning tree module. After the reduction of the maximum degree of the spanning tree, nodes can perform improvements using a faulty maximum degree information, but a spanning tree is always preserved. After a finite number of steps every node knows the correct maximum degree of the tree (see Lemma 4). Moreover,
Theorem 2 states that starting from an arbitrary configuration a spanning tree with degree at most $\Delta^* + 1$ is returned by the algorithm, with $\Delta^* + 1$ the degree of an optimal solution. Therefore, when each layer is stabilized the result follows. □

**Lemma 10 (closure).** Starting from a legitimate configuration, the algorithm preserves the legitimate configuration.

**Proof.** According to Lemmas 2 and 4, a spanning tree and its maximum degree are preserved. Now, we must show that starting from a configuration which describes a spanning tree with degree at most $\Delta^* + 1$ no improvement is performed by the algorithm, with $\Delta^* + 1$ the degree of an optimal solution. Theorem 1 states that when a minimum degree spanning tree with degree at most $\Delta^* + 1$ is reached, this implies there is no possible improvement for a spanning tree. In a legitimate configuration, the degree of every node from any non-tree edge is at least equal to the maximum degree minus one. Thus, the algorithm cannot perform an improvement in any fundamental cycle because the improving edge condition is not satisfied, either via procedure **Improve** or **Deblock**, (see Figure 3, lines 13 and 20). Therefore, starting from a legitimate configuration the algorithm preserves the legitimate configuration. □

6. Complexity issues

The following lemma states that the time complexity is polynomial while the memory complexity is logarithmic in the size of the network.

**Lemma 11.** Our algorithm converges to a legitimate state for the MDST problem in $O(mn^2 \log n)$ rounds using $O(\delta \log n)$ bits memory per node in the send/receive model\(^6\), where $\delta$ is the maximal degree of the network.

**Proof.** The algorithm is the composition of three layers: the spanning tree construction, the maximum degree computation and the tree degree reduction layer. The first and the second layer have respectively a time complexity of $O(n^2)$ rounds [4, 10] and $O(n)$ rounds (the proposed rules perform propagations of information in the spanning tree, as described in [17]). The time complexity is dominated by the last layer. According to [13], there are $O(n \log n)$ phases. A phase consists of decreasing by one the maximum degree in the tree (except for the last phase). In fact, let $k$ be the degree of the initial tree, there are $O(n/k)$ maximum degree nodes. As $2 \leq k \leq n - 1$, summing up the harmonic series corresponding to the $k$ values leads to $O(n \log n)$ phases. During each phase a non tree edge $\{u, v\}$ performs the following operations: finding its fundamental cycle and achieving an improvement (if it is an improving edge). To find its fundamental cycle, $\{u, v\}$ uses the procedure **CycleSearch** which propagates a **Search** message via a DFS traversal of the current tree. So the first operation requires at most $O(n)$ rounds. For the second operation there are two possible cases: a direct and an indirect improvement. A direct improvement is performed if $\{u, v\}$ is an improving edge, in this case a **Remove** message is sent and follows the fundamental cycle to make the different changes, requiring $O(n)$ rounds (see Lemmas 5, and 8). An indirect improvement is performed when $u$ or $v$ are blocking node ($\{u, v\}$ is a blocking edge), see Lemma 6. In the worst case, the blocking edges chain is of size at most $m - (n - 1)$. Therefore $m - (n - 1)$ exchanges are necessary to reduce the tree degree. This requires $O(mn)$ rounds (according to the direct improvement). So as the second operation needs at most $O(mn)$ rounds, the third layer requires $O(mn^2 \log n)$ rounds.

In the following we analyze the memory complexity of our solution. Each node maintains a constant number of local variables of size $O(\log n)$ bits. However, due to specificity of the model we use (the send/receive model) each node maintains a local copy of the states of its neighbors. Since $O(\log n)$ bits are necessary to

---

\(^6\)In the classical message passing model the memory complexity is $O(\log n)$. 

17
store the local variables of a node, the memory complexity per node including the copies of the neighborhood local variables is $O(\delta \log n)$ bits, where $\delta$ is the maximal degree of the network.

Note that the messages exchanged during the execution of our algorithm carry information related to the size of fundamental cycles. Therefore, the buffers length complexity of our solution is $O(n \log n)$.

One can advocate for an alternative implementation of the minimum-degree spanning tree. That is, consider a unique node collects the information to compute the minimal degree. This can be implemented if for example the root of a spanning tree of the network collects all the information related to the topology. Note that this solution may seem appealing by its simplicity however it has some disadvantages. The time complexity is still polynomial ($O(n^2)$ due to the construction of the spanning tree) while the memory complexity is multiplied by a factor of $n$ with respect to the solution we proposed since the root of the tree needs to store the local states of every node in the network. This solution is not scalable since at each join/leave the root should be informed and proceed to a new computation of the minimum degree which induces an important communication overhead.

**Theorem 3.** Our algorithm is a self-stabilizing distributed algorithm computing a spanning tree with degree at most $\Delta^* + 1$ in $O(mn^2 \log n)$ rounds and using $O(\delta \log n)$ bits memory per node in the send/receive model, with $\Delta^*, \delta, m$ and $n$ are respectively the minimum possible degree of a spanning tree, the maximal degree, the edges number and the nodes number of the network.

### 7. Conclusion and open problems

We proposed a self-stabilizing algorithm for constructing a minimum-degree spanning tree in undirected networks. Starting from an arbitrary state, our algorithm is guaranteed to converge to a legitimate state describing a spanning tree whose maximum node degree is at most $\Delta^* + 1$, where $\Delta^*$ is the minimum possible maximum degree of a spanning tree of the network.

To the best of our knowledge our algorithm is the first self-stabilizing solution for the construction of a minimum-degree spanning tree in undirected graphs. The algorithm can be easily adapted to large scale systems since it uses only local communications and no node centralizes the computation of the MDST. That is, each node exchanges messages with its neighbors at one hop distance and the computations are totally distributed. Additionally the algorithm is designed to work in any asynchronous message passing network.

The time complexity of our solution is $O(mn^2 \log n)$ where $n$ and $m$ are respectively the number of nodes and edges in the network. The memory complexity is $O(\delta \log n)$ in the send/receive atomicity model.

Several problems remain open. First, the computation of a minimum-degree spanning tree in directed topologies seems to be a promising research direction with a broad area of application (i.e., sensor networks, ad-hoc network, robot networks). Second, the new emergent networks need scalable solutions able to cope with nodes churn. An appealing research direction would be to design a super-stabilizing algorithm [3] for approximating a MDST. Finally, a distributed algorithm was proposed by Lavault et al. in [19] resolving the following bicriteria problem: Construct a minimum spanning tree of maximum degree at most $b\Delta^* + \log_b n$, for any constant $b > 1$, where $\Delta^*$ is the maximum degree of an optimal solution. It would be interesting to design a self-stabilizing algorithm considering the above bicriteria problem.

An interesting research direction from the engineering point of view would be to implement the proposed solution using lighter communication. However, due to the stabilization requirements there is an important trade-off between the accuracy with which nodes detect an inconsistency in their neighbourhood and the frequency of the state related messages are exchanged between neighbours.

### References


Springer (Ed.), 4th International Workshop on Distributed Algorithm (WDAG), Vol. LNCS 486, 1991,
pp. 15–28.


[8] F. Butelle, C. Lavault, M. Bui, A uniform self-stabilizing minimum diameter tree algorithm (extended

[9] L. Higham, Z. Liang, Self-stabilizing minimum spanning tree construction on message-passing networks,

(October 2003).


[12] M. Fürrer, B. Raghavachari, Approximating the minimum degree spanning tree to within one from the

[13] M. Fürrer, B. Raghavachari, Approximating the minimum-degree steiner tree to within one of optimal,


International Conference on Distributed Computing Systems (ICDCS), 2007, p. 27.


[19] C. Lavault, M. Valencia-Pabon, A distributed approximation algorithm for the minimum degree mini-