A distributed approximation algorithm for the minimum degree minimum weight spanning trees

Christian Lavault, Mario Valencia-Pabon

A distributed approximation algorithm for the minimum degree minimum weight spanning trees

Christian Lavault[1] and Mario Valencia-Pabon[2]

[1] LIPN, CNRS UMR 7030, Université Paris-Nord
Av. J.-B. Clément, 93430 Villetaneuse, France
email: lavault@lipn.univ-paris13.fr
[2] Departamento de Matemáticas, Universidad de los Andes
Cra. 1 No. 18A - 70, Bogotá, Colombia
email: mvalenci@uniandes.edu.co

Abstract. Fischer [3] has shown how to compute a minimum weight spanning tree of degree at most $b\Delta^* + \lceil \log_b n \rceil$ in time $O(n^{4+1/\ln b})$ for any constant $b > 1$, where Δ^* is the value of an optimal solution and n is the number of nodes in the network. In this paper, we propose a distributed version of Fischer's algorithm that requires messages and time complexity $O(n^{2+1/\ln b})$, and $O(n)$ space per node.

Keywords: distributed algorithms, approximation algorithms, minimum degree minimum weight spanning trees.

1 Introduction

Many computer communications networks require nodes to broadcast information to other nodes for network control purposes; this is done efficiently by sending messages over a spanning tree of the network. Distributed minimum weight spanning tree algorithms are useful in communication networks when one wishes to broadcast information from one node to all other nodes and there with one cost assigned to each channel in the network. If in such a minimum weight spanning tree the degree of a node is large, it might cause an undesirable communication load in that node. Therefore, the construction of minimum weight spanning trees in which the degree of a node is the lowest possible is needed. While it is easy enough to optimize the weight of a spanning tree, it is often more difficult to satisfy constraints which also involve the degrees of the nodes. The problem of minimizing the maximum degree of a spanning tree is known to be NP-hard, as the Hamiltonian path problem is merely a special case of this problem [6]. In this paper, we consider the problem of finding a distributed approximation algorithm for finding a minimum weight spanning tree whose maximum degree is as low as possible.

Previous and Related Work. Let Δ^* be the degree of an optimal solution. When edge weights are not considered, or assumed uniform, a $\Delta^* + 1$ approximation algorithm for minimizing the degree of spanning trees has been obtained by Fürer and Raghavachari [4]. A distributed version of the algorithm of Fürer an

Raghavachari which maintains the same performance approximation guarantee is proposed by Blin and Butelle [2]. For the weighted case, Fischer [3] gives an approximation algorithm that computes a minimum weight spanning tree of degree at most $b\Delta^* + \lceil \log_b n \rceil$ in time $O(n^{4+1/\ln b})$ for any constant $b > 1$, which is the best-known algorithm for this problem up to now. His algorithm is an adaptation of a local search algorithm of Fürer and Raghavachari [4] to the weighted case. Recently, Neumann and Laumanns in [7] extend Fischer's algorithm to spanning forests.

Our Results. In this paper, we propose a distributed version of Fischer's approximation algorithm that computes a minimum weight spanning tree of degree at most $b\Delta^* + \lceil \log_b n \rceil$, for any constant $b > 1$, where n is the number of nodes of the network and Δ^* is the maximum degree value of an optimal solution. Our distributed algorithm requires $O(n^{2+1/\ln b})$ messages and (virtual) time, and $O(n)$ space per node. From the complexity analysis of our distributed algorithm, we are able to derive that Fischer's sequential algorithm can be performed in $O(n^{3+1/\ln b})$ time, which improves on Fischer's upper bound on the time complexity.

The paper is organized as follows. In Section 2, we introduce the model of computation and we present Fischer's sequential algorithm to compute a minimum degree minimum weight spanning tree. In Section 3 we describe our distributed algorithm and in Section 4 we prove its correctness and complexity. Finally, Section 5 provides some concluding remarks.

2 Preliminaries

The model. We consider the standard model of asynchronous static distributed system. The point-to-point communication network is associated a weighted undirected graph $G = (V, E, w)$. The set of nodes V represents the processors of the network, the set of edges E represents bidirectional non-interfering communication channels operating between neighboring nodes, and w is a real-valued function defined on E, which represents a cost assigned to each channel of the network. No common memory is shared by the nodes (processes). In such networks, any process can generate one single message at a time and can send it to all its neighbors in one time step. Every process owns one distinct identity from $\{1, \ldots, n\}$. However, no node has a global knowledge of the network topology, except of its own incident edges (e.g., every node is unaware of its neighbors' identities). The distributed algorithm is event-driven and does not use time-outs, i.e. nodes can not access a global clock in order to decide what to do. Moreover, each node runs the algorithm, determining its response according to every type of message received. Namely, the algorithm specifies for any one node which computation is to be performed and/or which message be sent. The algorithm is started independently by all nodes, perhaps at different times. When the algorithm starts, each node is unaware of the global network topology but of its own edges. Upon termination, every node knows its neighbors' identities within an approximated minimum degree minimum weight spanning tree. The

efficiency of a distributed algorithm is evaluated in terms of *message*, *time* and *space* complexity as follows (see [8]). The *message complexity* of a distributed algorithm is the total number of messages sent over the edges. We also assume that each message contains $O(\log n + R)$ bits, where $|V| = n$ and R is the number of bits required to represent any real edge weight. In practical applications, messages of such a kind are considered of "constant" size. The *time complexity* is the total (normalized) time elapsed from a change. The *space complexity* is the space usage per node.

The Problem. Let $G = (V, E, w)$ be a real-weighted graph modeling the communication network. A spanning tree $T = (V_T, E_T)$ of G is a tree such that $V_T = V$ and $E_T \subseteq E$. The weight of a spanning tree T of G equals the sum of the weights of the $|V| - 1$ edges contained in T, and T is called a *minimum weight spanning tree*, or MWST, if no tree has a smaller (minimum) weight that T. Our goal is to find a distributed polynomial algorithm to compute a MWST such that its maximum degree is as low as possible. We denote by $N_G(u)$ and $N_T(u)$ the set of neighbors of node u in G and in T, respectively.

Fischer's sequential algorithm. Let T be a tree on n nodes. Define the **rank** of T to be the ordered n-tuple $(t_n, \ldots, t_1)$ where t_i denotes the number of nodes of degree i in T. Define a lexicographical order on these ranks; a tree T' on n nodes is of lower rank that T if $t'_j < t_j$ for some j and $t'_i = t_i$ for $i = j+1, \ldots, n$. Clearly, when an edge is added to a spanning tree, it creates a cycle. Conversely, removing any edge from the induced cycle results again in a spanning tree. A *swap* is defined to be any such exchange of edges; a swap is said *cost-neutral* if the edges exchanged are of equal weight. Consider a swap between the edges $xw \in T$ and $uv \notin T$, with $x \notin \{u, v\}$. Such a swap may increase by one the degree of both u and v in T, but it also reduces the degree of x. So, the rank of T is decreasing if the degree of x in T is at least the maximal degree of u and v plus 2. A **locally optimal** minimum weight spanning tree is a MWST in which no cost-neutral swap can decrease the rank of the tree.

Theorem 1. *(Fischer [3]) If T is a locally optimal MWST, and Δ_T is the maximum degree in T, then $\Delta_T \leq b\Delta^* + \lceil \log_b n \rceil$ for any constant $b > 1$, where Δ^* is the maximum degree of an optimal solution.*

As can be deduced from the proof of Theorem 1 in Fischer's paper [3], in order to construct a locally optimal spanning tree, it is sufficient to consider those nodes with degree at least equal to $\Delta_T - \lceil \log_b n \rceil$ among high-degree nodes. Fischer's polynomial algorithm to compute a locally optimal spanning tree can be described as follows.

Fischer's algorithm :

0. Start with any MWST T. Let $b > 1$ be the desired approximation parameter. Let $l < n$ be the number of distinct edge weights, $w_1, \ldots, w_l$, in T.
1. Let Δ_T be the current maximum degree in T.
2. For every node $v \in G$, check for appropriate improvements. Conduct a depth first traversal of T starting from v.

2.1. Let w be the current vertex on the traversal of T, and P be the vw-path in T.
2.2. Assign variables $M_1, \ldots, M_l$ such that M_i denotes the maximum degree of those nodes adjacent to edges of weight w_i in P.
2.3. If there is an edge $vw \in G$, let w_i be its weight. If M_i is at least two greater that the degree of v and w in T, and $M_i \geq \Delta_T - \lceil \log_b n \rceil$, the the edge uv can be used to reduce the high-degree rank of T. Conduct the appropriate swap on T, and repeat to Step (1) for the next round.
2.4. If no appropriate cost-neutral swap was found in any of the traversals, terminate.

Fischer proves in [3] that each round of his previous algorithm takes $O(n^2)$ time and that the number of rounds can be bounded by $O(n^{2+1/\ln b})$. Therefore, Fischer's algorithm computes a locally optimal minimum weight spanning tree in time $O(n^{4+1/\ln b})$.

3 Description of the algorithm

In Section 3.1, a high-level description of our distributed algorithm is given, and in Section 3.2, we detail the description of the algorithm. Every process is running the following procedure, which consists of a list of the responses to each type of messages generated. Each node is assumed to queue the incoming messages and to reply them in First-Come, First-Served order (FIFO). Any reply sent is completed before the next is started and all incoming messages are also delivered to each node via an initially empty First-Come First-Served queue.

3.1 High-Level Description

Let G be a connected graph modeling an interconnection network. We now describe the general method used to construct a locally optimal minimum weight spanning tree T of G. First, we assume that some current minimum weight spanning tree T of G is already constructed. (Various MWST distributed algorithms are $\Theta(|E| + n \log n)$ message optimal, e.g. [1,5], while the best time complexity achieved is $O(n)$ in [1].) Next, for each edge $pr \in T$, let T_r (resp. T_p) be the subtree of $T \setminus pr$ containing the node r (resp. p) (see Fig. 1). The algorithm is divided into rounds, and each round is consisting of the following four phases.

• **Initialization phase.** Let Δ_T be the maximum current degree of the MWST T. First, each node of T must know Δ_T. This can be done by starting an election on T, where all nodes broadcast their degree in T. Before checking an edge to find an appropriate cost-neutral swap (see Fig. 1), we need to do the following. For each edge pr in T and for each node $x \in T$, we determine, which of the nodes p or r is the closer to x (in terms of distance in T). For the purpose, the node p initiates a broadcast by forwarding the token $\langle pr \rangle$ to all nodes in T_p. When a node $x \in T_p$ receives the token $\langle pr \rangle$, x knows that for any appropriate swap

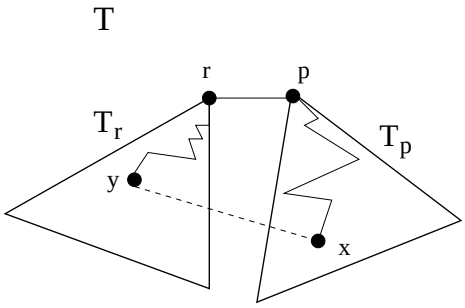

Fig. 1. Example of a cost-neutral swap between edges $pr \in T$ and $xy \notin T$.

involving the edge pr, it needs to find a neighbor y in G such that $y \in T_r$. In an analogous way, node r initiates a similar broadcast in T_r.

• **Search-edge phase.** Whenever an edge pr in the current MWST T ends the Initialization phase, it begins searching for an edge $xy \notin T$ to perform an appropriate swap. Thus, node p starts a search for a node $x \in T_p$ with a neighbor $y \in T_r$, for which $w(pr) = w(xy)$ and such that the maximal value of the degrees of p and r in T is at least two greater than the degree of x and y. If such a node x is found, it sends a message to p (via edges in T_p) to announce success. If there is no node in T_p meeting the previous conditions then p is informed that edge pr can not be used to an appropriate swap on T. Similarly, node r starts such a search in T_r. If no appropriate cost-neutral swap is found in this phase, the algorithm terminates.

• **Edge-election phase.** If a node p enters this phase then, there exists a pair of edges $pr \in T$ and $xy \notin T$ which can execute an appropriate swap on T. However, p must ensure that it belongs to the one only pair of edges that can achieve such a swap. So, p starts an election procedure and it chooses among the possible initiators (i.e. the nodes that reached this phase) one node to perform the swap. So, each initiator p forwards its identity to all other nodes in T. The elected node is the one with the minimal identity.

• **Edge-exchange phase.** If node p wins the Edge-election phase, then there exists again at least one pair of edges $pr \in T$ and $xy \notin T$ that can be used to complete the appropriate swap on T (reducing the high-degree rank of T). Thus, p informs r that they are not connected in T anymore and starts a search in T_p for the node x (assuming $x \in T_p$). When x is found, it informs y that they are now connected within T. Finally, x sends a message to all other nodes in the current tree to inform that a new round can initiate.

3.2 Detailed description

The algorithm is described for a node p in the network. We begin by describing the variables maintained by p.

Local variables of p: (Let T denote the current minimum weight spanning tree.)

- $Neigh_p[r]$: $\{branch,\ unbranch\}$. Node p maintains the variable for each $r \in N_G(p)$. An edge pr belongs to T if and only if $Neigh_p[r] = branch$.
- $d_p^T, \Delta_T, d_p[r]$: *integer*; $w_p[r]$: *real*. The variables d_p^T and Δ_T denote the degree of p and the maximal degree of T, respectively. For each $r \in N_G(p)$, the variables $d_p[r]$ and $w_p[r]$ denote the degree of the neighbor r of p in T and the weight of the edge pr in G, respectively.
- $Father_p[r]$: *integer*. For each edge $ry \in T$, the variable $Father_p[r]$ denotes the neighbor of p closer to node r in T_r (if $p \in T_r$). *Default value:* $Father_p[r] = p$, for each $r \in T$.
- $Side_p[uv]$: *integer*. For each edge $uv \in T$, the variable $Side_p[uv]$ denotes the node $r \in \{u, v\}$ closer to p in T. *Default value:* $Side_p[uv] = udef$, for each edge $uv \in T$.
- $CountSide_p[uv], CountFail_p[uv]$: *integer*. For each edge $uv \in T$, the variables $CountSide_p[uv]$ and $CountFail_p[uv]$ maintain the number of neighbors of p which already instanced the variable $Side\_[uv]$. Moreover, the variable $CountFail_p[uv]$ counts the number of neighbors of p which find no edge to replace uv. *Default values:* $CountSide_p[uv] = CountFail_p[uv] = 0$, for each edge $uv \in T$.
- $EndInit_p[r] : \{0, 1, 2\}$. For each $r \in N_T(p)$, the variable maintains the state of the Initialization phase of the edge $pr \in T$. *Default value:* $EndInit_p[r] = 0$, for each $r \in N_T(p)$.
- $EdgeFind_p[uv] : \{0, 1\}$. For each edge $uv \in T$, the boolean variable is used to check whether an edge $xy \notin T$ is found in G which replaces uv. *Default value:* $EdgeFind_p[uv] = 0$, for each edge $uv \in T$.
- $NeighFail_p, \#Fails_p$: *integer*. The variable $NeighFail_p$ maintains the number of neighbors of p in T s.t. the associated edge in T is useless to any edge-exchange in a current round. If $NeighFail_p = |N_T(p)|$, it means that no edge incident to p can be used in view of an exchange in a current round. The variable $\#Fails_p$ maintains the number of nodes in T for which there is no edge incident to the ones that can be used in an exchange. If in a round, $\#Fails_p = |T|$, it means that the algorithm is terminated. *Default values:* $NeighFail_p = \#Fails_p = 0$.
- $Mode_p$: $\{election,\ non\text{-}election\}$. If during a Search-edge phase a subset of edges in T finds an edge appropriate to an exchange, all corresponding edges-incident nodes begin an election to decide on the edge to be exchanged. If p knows that an Election phase is running, $Mode_p = election$. *Default value:* $Mode_p = non\text{-}election$.
- $State_p$: $\{winner,\ loser,\ udef\}$. The variable maintains the state of p during an Edge-election phase (*winner or loser*). If $State_p = udef$, it means that p does not know if an Edge-election phase is taking place in a current round. *Default value:* $State_p = udef$.
- $CountLoser_p$: *integer*. During an Edge-election phase, the variable maintains the number of nodes that lost the election w.r.t. node p. Clearly, if

$CountLoser_p = |T| - 1$, node p wins the election, i.e. an edge incident to p becomes elected. *Default value:* $CountLoser_p = 0$.

- $NodeElec_p$: *integer*. The variable maintains the identity of the winning node during an Edge-election phase. *Default value:* $NodeElec_p = udef$.
- $EdgeElec_p$: *integer*$\times$ *integer*$\times$ *integer*$\times$ *integer*$\times$ *integer*. The variable maintains the pair of elected edges (if any) in view of an appropriate exchange in a current round. If $EdgeElec_p = (d, p, r, x, y)$, then a pair of edges pr and xy is found in G, s.t. $pr \in T$, $xy \notin T$, $Side_x[pr] = p$, $Side_y[pr] = r$, $w_x[y] = w_p[r]$, and where $d = \max\{d_p^T, d_p[r]\}$. *Default value:* $EdgeElec_p = udef$.

Now, each node p executes the following steps.

Initial assumptions: Assume that the algorithm starts with any MWST T of G already given. We need the algorithm constructing T to be "process terminating" (i.e. every node knows that the MWST algorithm is terminated; however, no distributed termination detection is available). So, we can assume that p knows its degree d_p^T in T, and for each $r \in N_G(p)$, the variables $Neigh_p[r]$, $d_p[r]$ and $w_p[r]$ are correctly computed. Let $b > 1$ be the desired approximation parameter.

Initialization phase :

1. In order to determine the maximum degree Δ_T in the current tree T, p initiates an election (all nodes in T are initiators) by sending the token $\langle d_p^T, p\rangle$ to all its neighbors in T. Node p wins the election if the token $\langle d_p^T, p\rangle$ is maximal w.r.t. the lexicographic order. As all vertices are initiators, we also use this step to initialize the remaining local variables of p with their default values (see the definition of local variables of p above).
2. For each edge $uv \in T$ and for each node $r \in T$, we need to compute the values of variables $Side_p[uv]$ and $Father_p[r]$. This is done as follows.
   2.1. **For** each $z \in N_T(p)$ **do**
   - $Side_p[pz] \leftarrow p$;
   - send $\langle$**side**$, p, z, p\rangle$ to each node $r \in N_T(p)$ s.t. $r \neq z$;

   2.2. Upon receipt of $\langle$**side**$, u, v, q\rangle$
   - $Side_p[uv] \leftarrow u$;
   - $Father_p[u] \leftarrow q$;
   - **If** p is not a leaf **then** send $\langle$**side**$, u, v, p\rangle$ to each node $r \in N_T(p)$ s.t. $r \neq q$;
   - **If** p is a leaf **then** send $\langle$**end-side**$, u, v\rangle$ to node $Father_p[u]$;

   2.3. Upon receipt of $\langle$**end-side**$, u, v\rangle$
   - $CountSide_p[uv] \leftarrow CountSide_p[uv] + 1$;
   - **If** $p \neq u$ **and** $CountSide_p[uv] = |N_T(p)| - 1$ **then** send $\langle$**end-side**$, u, v\rangle$ to node $Father_p[u]$;
   - **If** $p = u$ **and** $CountSide_p[uv] = |N_T(p)| - 1$ **then**
     - send $\langle$**check-side**$, u\rangle$ to node v;
     - $EndInit_p[v] \leftarrow EndInit_p[v] + 1$;
     - **If** $EndInit_p[v] = 2$ **then** edge $pv \in T$ finishes the Initialization phase and can **go to** Step 3.

2.4. Upon receipt of $\langle$**check-side**, $q\rangle$
- $EndInit_p[q] \leftarrow EndInit_p[q] + 1$;
- **If** $EndInit_p[q] = 2$ **then** edge $pq \in T$ finishes the Initialization phase and can **go to** Step 3.

Search-edge phase:

3. If there is some neighbor q of p in T for which $EndInit_p[q] = 2$, it means that the edge $pq \in T$ is ready to initiate a search for some possible unused edge xy of G in view of an appropriate exchange. For our purpose, the edge $pq \in T$ must meet the condition $\max\{d_p^T, d_p[q]\} \geq \Delta_T - \lceil \log_b n \rceil$. Otherwise, pq can not be considered in this phase. This is done as follows.

3.1. **If** $\exists q \in N_T(p)$ s.t. $EndInit_p[q] = 2$ **and** $\max\{d_p^T, d_p[q]\} \geq \Delta_T - \lceil \log_b n \rceil$ **then** send $\langle$**change**$, p, q, d_{pq}, w_p[q]\rangle$ to each node $r \in N_T(p)$, with $r \neq q$, and where $d_{pq} = \max\{d_p^T, d_p[q]\}$.

3.2. Upon receipt of $\langle$**change**$, u, v, d_{uv}, w_u[v]\rangle$
- **If** $\exists r \in N_G(p)$ s.t. $Neigh_p[r] = unbranch, Side_r[uv] = v, w_p[r] = w_u[v]$, and $d_{uv} \geq \max\{d_p^T, d_p[r]\} + 2$ **then** send $\langle$**find**$, u, v, p, r\rangle$ to node $Father_p[u]$;
  else
  - **If** p is a leaf **then** send $\langle$**fail**$, u, v\rangle$ to node $Father_p[u]$;
  - **If** p is not a leaf **then** send $\langle$**change**$, u, v, d_{uv}, w_u[v]\rangle$ to each node $r \in N_T(p)$ s.t. $r \neq Father_p[u]$;

3.3. Upon receipt of $\langle$**find**$, u, v, q, r\rangle$
- $EdgeFind_p[uv] \leftarrow 1$;
- **If** $p \neq u$ **then** send $\langle$**find**$, u, v, q, r\rangle$ to node $Father_p[u]$;
- **If** $p = u$ **then** p is ready to begin the Edge-election phase and can **go to** Step 4.

3.4. Upon receipt of $\langle$**fail**$, u, v\rangle$
- $CountFail_p[uv] \leftarrow CountFail_p[uv] + 1$;
- **If** $EdgeFind_p[uv] \neq 1$ **then**
  - **If** $p \neq u$ **and** $CountFail_p[uv] = |N_T(p)| - 1$ **then** send $\langle$**fail**$, u, v\rangle$ to node $Father_p[u]$;
  - **If** $p = u$ **and** $CountFail_p[uv] = |N_T(p)| - 1$ **then**
    $NeighFail_p \leftarrow NeighFail_p + 1$;
    send $\langle$**fail1**$, u, v\rangle$ to node v;
    If $NeighFail_p = |N_T(p)|$ **then** send $\langle$**end**$, p\rangle$ to each node $r \in N_T(p)$ in order to check whether the algorithm is terminated;

3.5. Upon receipt of $\langle$**fail1**$, q, p\rangle$ or $\langle$**end**$, r\rangle$
- $\#Fails_p \leftarrow \#Fails_p + 1$;
- **If** $\#Fails_p = |T|$ **then** the algorithm terminates.

Edge-election phase:

4. This phase begins when p receives a message $\langle$**find**$, p, r, x, y\rangle$, which means that the edge $pr \in T$ can be exchanged for an unused edge $xy \in G$. Before

the change, p must ensure that it is the only node in T that performs this operation. To this end, p initiates an election procedure in T and it picks among the possible initiators (i.e. the nodes that reached this phase) the one that makes the edge exchange. Note that from this step till the end of the current round, any message of type $\langle$**side**$\rangle$, $\langle$**end-side**$\rangle$, $\langle$**check-side**$\rangle$, $\langle$**change**$\rangle$, $\langle$**find**$\rangle$, $\langle$**fail**$\rangle$, $\langle$**fail1**$\rangle$, and $\langle$**end**$\rangle$ received by p is ignored. Let $d_{pr} = \max\{d_p^T, d_p[r]\}$.

4.1. **If** $Mode_p = non\text{-}election$ **then**
- $EdgeElec_p \leftarrow (d_{pr}, p, r, x, y)$; $NodeElec_p \leftarrow p$; $Mode_p \leftarrow election$;
- Send $\langle$**elec**$, p, p\rangle$ to each node $q \in N_T(p)$;

else ($Mode_p = election$)
- **If** $NodeElec_p = p$ **and** $State_p = udef$ **then**
  let $EdgeElec_p = (d, p, z, a, b)$; **If** $d_{pr} > d$ **then** $EdgeElec_p \leftarrow (d_{pr}, p, r, x, y)$;

4.2. Upon receipt of $\langle$**elec**$, q, z\rangle$
- **If** $Mode_p = non\text{-}election$ **then**
  - $NodeElec_p \leftarrow q$; $Mode_p \leftarrow election$; $State_p \leftarrow loser$;
  - Send $\langle$**lost**$, p, p\rangle$ to each node $r \in N_T(p)$;
  - **If** p is not a leaf **then** send $\langle$**elec**$, q, p\rangle$ to each node $r \in N_T(p)$ s.t $r \neq z$;

  else ($Mode_p = election$)
  - **If** $q < NodeElec_p$ **then**
    - **If** $State_p \neq loser$ and $NodeElec_p = p$ then
      $State_p \leftarrow loser$;
      send $\langle$**lost**$, p, p\rangle$ to each node $r \in N_T(p)$;
    - **If** p is not a leaf **then** send $\langle$**elec**$, q, p\rangle$ to each node $r \in N_T(p)$ s.t. $r \neq z$;
    - $NodeElec_p \leftarrow q$;

4.3. Upon receipt of $\langle$**lost**$, q, t\rangle$
- **If** $Mode_p = election$ **and** $State_p = udef$ **then**
  - **If** $p \neq q$ **then** $CountLoser_p \leftarrow CountLoser_p + 1$.
  Moreover, if $CountLoser_p = |T| - 1$ **then** $State_p \leftarrow winner$; **go to** Step 5.
  - **If** $p = q$ **then** $State_p \leftarrow loser$;
- **If** $State_p \neq winner$ **and** p is not a leaf **then** send $\langle$**lost**$, q, p\rangle$ to each node $r \in N_T(p)$ s.t. $r \neq t$.

Edge-exchange phase:

5. If p wins the Edge-election phase, i.e. $State_p = winner$, with $EdgeElec_p = (d_{pr}, p, r, x, y)$. Then the edge $pr \in T$ is ready to be exchanged for the unused edge $xy \in G$. Note that from this step till the end of the current round, any message received by p which is distinct from types $\langle$**disconnect**$\rangle$, $\langle$**connect**$\rangle$, $\langle$**new**$\rangle$, and $\langle$**nround**$\rangle$ is ignored. So, during this phase the following is done.

5.1. - Send $\langle$**disconnect**$, r, p\rangle$ to node r;
- $Neigh_p[r] \leftarrow unbranch$; $d_p[r] \leftarrow d_p[r] - 1$; $d_p^T \leftarrow d_p^T - 1$;
- Send $\langle$**connect**$, p, r, x, y\rangle$ to each node $t \in N_T(p) : t \neq r$;

5.2. Upon receipt of $\langle \textbf{disconnect}, p, q \rangle$
- $Neigh_p[q] \leftarrow unbranch$; $d_p[q] \leftarrow d_p[q] - 1$; $d_p^T \leftarrow d_p^T - 1$;

5.3. Upon receipt of $\langle \textbf{connect}, u, v, x, y \rangle$
- **If** $p \neq x$ and p is not a leaf **then** send $\langle \textbf{connect}, u, v, x, y \rangle$ to each node $t \in N_T(p) : t \neq Father_p[u]$;
- **If** $p = x$ **then**
  - send $\langle \textbf{new}, y, p \rangle$ to node y (via the unused edge $py \in G$);
  - $Neigh_p[y] \leftarrow branch$; $d_p[y] \leftarrow d_p[y] + 1$; $d_p^T \leftarrow d_p^T + 1$;

5.4. Upon receipt of $\langle \textbf{new}, p, q \rangle$
- $Neigh_p[q] \leftarrow branch$; $d_p[q] \leftarrow d_p[q] + 1$; $d_p^T \leftarrow d_p^T + 1$;
- send $\langle \textbf{nround}, p \rangle$ to each node $t \in N_T(p)$;

5.5. Upon receipt of $\langle \textbf{nround}, z \rangle$
- **If** p is not a leaf **then** send $\langle \textbf{nround}, p \rangle$ to each node $w \in N_T(p)$ s.t. $w \neq z$;
- **Go to** step 1. (This round is ended and p executes a new round.)

4 Correctness and Complexity

Theorem 2. *The distributed algorithm described in Section 3 for computing a locally optimal minimum weight spanning tree is correct and deadlock-free.*

Proof. Let T be the MWST computed in a current round of the algorithm. If any node p of T meets the condition $\#\text{Fails}_p = |T|$ (the number of nodes in T), it means that no edge in T can find another edge outside T in view of a cost-neutral swap to reduce the rank of T. In such a situation, the algorithm terminates. By definition, T is indeed a locally optimal minimum weight spanning tree and the algorithm is correct. Moreover, by construction, when a node of T generates a message to send through the edges in T, it uses each edge of T at most once; and it is removed (i.e. disappears) either by reaching its destination or whenever it arrives to a leaf of T. Upon receipt of a message, any node p executes some instructions if p and the information within the message meet specific properties. Otherwise, the message is ignored by the node. Finally, when a cost-neutral swap reducing the rank of T is found, all nodes in T are informed and initiate a new round (see Steps 5.4 and 5.5). Otherwise, all nodes in T are informed that the algorithm is terminated (see Steps 3.5 and 3.6). Consequently, we may conclude that the algorithm is also deadlock-free. □

Lemma 1. *Each round of the distributed algorithm in Section 3 requires $O(n)$ messages and time $O(n)$.*

Proof. We assume that the algorithm starts with any MWST T of G available beforehand ($|T| = |V| = n$). In order to compute the maximum degree of the current tree T, an election is triggered in Step (1). The time and message complexity of such a tree election is $O(n)$ (see [8]). Now, consider any edge $uv \in T$. By construction, both nodes u and v generate one single message of each type

⟨**side**⟩, ⟨**change**⟩ and ⟨**connect**⟩ (see Steps 2.1, 3.1 and 5.1, resp.), which traverses only once each edge in T_u and T_v, respectively. These messages disappear whenever they reach a leaf node. Therefore, the number of such messages sent over T is at most equal to $3(n-2)$. Next, consider the edges of T that are incident to nodes in T_u. One of the nodes incident to such edges generates one message of each type ⟨**end-side**⟩, ⟨**find**⟩ and ⟨**fail**⟩ (see Steps 2.2 and 3.2, resp.), which is sent to node u. Thus, these messages traverse each edge in T_u only once and the same property holds in the subtree T_v. Therefore, the number of such messages sent over T is at most equal to $3(n-2)$. Further, each node in T can generate one message of each type ⟨**elec**⟩, ⟨**lost**⟩ and ⟨**nround**⟩ (see Steps 4.1, 4.2 and 5.4, resp.), which is broadcast through the tree T. Therefore, the number of such messages sent over T is at most equal to $3(n-1)$. Finally, The remaining messages are exchanged once between neighbors. Therefore, the number of messages sent over T within a current round is $O(n)$. The time complexity of the algorithm during a round can be derived from the above analysis by observing that a node u can broadcast a message to all its neighbors in one time step and the distance in T from u to any destination node is at most $n-1$. □

Lemma 2. *For any constant $b > 1$, the number of rounds used by the distributed algorithm in Section 3 can be bounded from above by $O(n^{1+1/\ln b})$.*

Proof. We use a potential function similar to Fischer's, with the difference that only high-degree nodes have a potential value greater or equal to one. A similar potential function is also used in [7]. Let Δ_T be the maximum degree of the current MWST T on n nodes during a round of the algorithm. Let $\delta = \max\{\Delta_T - \lceil \log_b n \rceil, 0\}$ and let d_v^T be the degree of a node v in T. The *potential* of the node v is define as follows,

$$\phi_v = \begin{cases} e^{d_v^T - \delta} & \text{if} \quad d_v^T \geq \Delta_T - \lceil \log_b n \rceil, \\ 1/2 & \text{otherwise.} \end{cases}$$

Denote $\Phi_T = \sum_{v \in T} \phi_v$. Since $b > 1$, $\Phi_T \leq n e^{\lceil \log_b n \rceil} \leq n e^2 n^{1/\ln b}$.
Let us now compute the change in potential, $\Delta\Phi$, when the algorithm performs a local improvement involving a node of degree at least $\Delta_T - \lceil \log_b n \rceil$ in T. Assume edge xy is added to T and edge uv is removed from T; also assume w.l.o.g. that $d_u^T \geq d_v^T$. Since the algorithm only performs swaps which reduce the degree of some high-degree node, we have that $d_u^T \geq \Delta_T - \lceil \log_b n \rceil$ and $d_u^T \geq \max\{d_x^T, d_y^T\} + 2$. By a simple case analysis, it is easy to check that, in a local improvement, the potential decreases of the smallest amount possible if $d_u^T = \Delta_T - \lceil \log_b n \rceil$ and $d_v^T, d_x^T, d_y^T < \Delta_T - \lceil \log_b n \rceil$. In such a case, any local improvement reduces the potential by $1/2$. Therefore, in any local improvement, $\Delta\Phi \geq 1/2$. This implies that after at most two local improvements (i.e. rounds), Φ_T decreases of at least one unit. Hence, the algorithm finds a locally optimal MWST in $O(n^{1+1/\ln b})$ rounds. □

Theorem 3. *For any constant $b > 1$, the distributed algorithm in Section 3 requires $O(n^{2+1/\ln b})$ message and time, and $O(n)$ space per node to compute a*

minimum weight spanning tree of maximum degree at most $b\Delta^* + \lceil \log_b n \rceil$*, where* n *is the number of nodes of the network and* Δ^* *is the maximum degree value of an optimal solution.*

Proof. The algorithm is assumed to start with any MWST T of G (e.g. by using the algorithm in [1] beforehand). Now, for each edge $uv \in T$, each node p maintains the variables $Side_p[uv]$, $CountSide_p[uv]$, $CountFail_p[uv]$ and $EdgeFind_p[uv]$ in particular (see Section 3.2). Since T is a tree on n nodes, its number of edges is $n-1$, and so the algorithm requires $O(n)$ space per node. Finally, by Theorems 1 and 2, and Lemmas 1 and 2, the theorem follows. □

Note that the proof of Lemma 2 works also in the sequential case. Therefore, we obtain the following corollary, which improves on Fischer's time complexity.

Corollary 1. *For any constant* $b > 1$*, Fischer's sequential algorithm in [3] (Section 2) finds a minimum weight spanning tree of maximum degree at most* $b\Delta^* + \lceil \log_b n \rceil$ *in* $O(n^{3+1/\ln b})$ *time.*

5 Concluding Remarks

In the paper, we present a distributed approximation algorithm which computes a minimum weight spanning tree of degree at most $b\Delta^* + \lceil \log_b n \rceil$, for any constant $b > 1$. The message and time complexity of the algorithm is $O(n^{2+1/\ln b})$ and it requires $O(n)$ space per node. To our knowledge, this is the first distributed approximation algorithm for the minimum degree minimum weight spanning tree problem.

References

1. B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems, *In Proc. Symp. on Theory of Computing*, 230-240, 1987.
2. L. Blin, F. Butelle. The first approximated distributed algorithm for the minimum degree spanning tree problem on general graphs, *Int. J. on Fond. in Comput. Sci.*, **15**(3): 507-516, 2004.
3. T. Fischer. Optimizing the degree of minimum weight spanning trees, *Technical Report 93-1338*, Department of Computer Science, Cornell University, Ithaca, NY, USA, 1993.
4. M. Fürer, B. Raghavachari. Approximating the minimum degree spanning tree to within one from the optimal degree, *Journal of Algorithms*, **17**:409-423, 1994 (a preliminary version appeared in SODA'92).
5. R. G. Gallager, P. A. Humblet, P. M. Spira. A distributed algorithm for minimum weight spanning trees, *ACM Trans. Program. Lang. Syst.*, **5**:67-77 (1983).
6. M. R. Garey, D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman Eds., San Francisco, 1979.
7. F. Neumann, M. Laumanns. Speeding up approximation algorithms for NP-hard spanning forest problems by multi-objective optimization, *Electronic Coll. on Comput. Complexity*, Report No 29, 2005.
8. G. Tel. *Introduction to Distributed Algorithms*, Cambridge University Press, 1994.

Appendix

In order to facilitate the reading of our paper, we give the proof of Fischer's Theorem 1 in Section 2 as follows.

Proof of Theorem 1 (Fischer [3]). Let $b > 1$ be any constant and let G be a connected graph on n nodes. Consider a locally optimal MWST T of G with maximum degree Δ_T. Let S_i denote the set of nodes of degree at least i in T. Clearly, $|S_{\Delta_T}| \geq 1$. Since $|S_i| \leq n$ for all i, the ratio $|S_{i-1}|/|S_i|$ can not be greater that b for $\log_b n$ consecutive values of i. Therefore, for any constant $b > 1$, there exists some integer δ in the range $\Delta_T - \lceil \log_b n \rceil \leq \delta \leq \Delta_T$ such that $|S_{\delta-1}|/|S_\delta| \leq b$. Suppose we choose an integer δ to satisfy this property, and remove from T the edges adjacent to nodes in S_δ. Let T_δ denote the remaining edges of T. As T is initially connected, then there are at least $\delta|S_\delta|+1-(|S_\delta|-1)$ or $(\delta-1)|S_\delta|+2$ connected components in T_δ.

Consider the graph G_δ formed by contracting every component of T_δ. Since any MWST of G must contain a MWST of G_δ, any MWST must include at least $(\delta-1)|S_\delta|+1$ edges from G_δ.

Consider an edge $vw \in G$ not in T between two components of T_δ. Let P^T denote the vw-path in T, and P^T_δ denote those edges of P^T which appear in G_δ, the edges on the path which are adjacent to any node in S_δ. Suppose neither v nor w is in $S_{\delta-1}$. Since T is locally optimal, no cost-neutral swap can reduce the rank of T, so vw must be more expensive that any edge in P^T_δ. Since vw and P^T_δ form a cycle in G_δ, this implies that vw may not participate in a MWST of G_δ. Therefore, only edges which are adjacent to nodes in $S_{\delta-1}$ may participate in a MWST of G_δ, and any MWST of G must contain at least $(\delta-1)|S_\delta|+1$ edges that are adjacent to $S_{\delta-1}$.

Earlier we chose δ to satisfy the inequality $|S_{\delta-1}|/b \leq |S_\delta|$. Substituting, we see there must be at least $((\delta-1)|S_{\delta-1}|/b)+1$ edges adjacent to nodes in $S_{\delta-1}$. Therefore, the average degree of a node in $S_{\delta-1}$ must be at least $\frac{(\delta-1)|S_{\delta-1}|+b}{b|S_{\delta-1}|}$, and so, $\Delta^* > \frac{\delta-1}{b}$.

Combining this with the possible range for δ, we find that $\Delta_T \leq b\Delta^* + \lceil \log_b n \rceil$. □