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

Christian Lavault[†] Mario Valencia-Pabon[†*]

[†] LIPN, CNRS UMR 7030 – {lavault,valencia}@lipn.univ-paris13.fr

Abstract

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

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 monitoring and 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, with one cost assigned to each channel of the network. If the degree of a node is large in such a minimum weight spanning tree (or MWST, defined in Section 2), it might cause an undesirable communication load at that node. In fact, there are instances, like the distribution of mail and news on the Internet, in which the broadcast needs not be executed on a priority basis. One of the parameters that different sites may want to reduce is the amount of work done by their site. Broadcasting information on a minimum degree minimum weight spanning tree (or MDMWST) is one such solution (more applications of the MDMWST problem can be found in [5]). 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

*Corresponding Author. LIPN, CNRS UMR 7030, Université Paris-Nord, Avenue J.-B. Clément, 93430 Villetaneuse, France. Email: valencia@lipn.univ-paris13.fr

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 [7]. In this paper, we consider the problem of finding a distributed approximation algorithm for finding a MWST whose maximum degree is as low as possible.

Let Δ^* be an optimal solution of the maximum degree value of any MWST for a given weighted graph. When edge weights are not considered, or assumed uniform, a $\Delta^* + 1$ sequential approximation algorithm for minimizing the maximum degree of spanning trees has been obtained by Fürer and Raghavachari [5]. A distributed version of the sequential algorithm of Fürer and Raghavachari which maintains the same performance approximation guarantee is proposed by Blin and Butelle [2]. For the weighted case, Fischer [4] gives a sequential approximation algorithm that computes a minimum weight spanning tree of maximum degree at most $b\Delta^* + \lceil \log_b n \rceil$ in time $\mathcal{O}\left(n^{4+1/\ln b}\right)$ for any constant $b > 1$, which is the best-known sequential approximation algorithm for this problem up to now. His algorithm is an adaptation of a local search algorithm of Fürer and Raghavachari [5] to the weighted case. Recently, Neumann and Laumanns in [8] extend Fischer's sequential algorithm to spanning forests. Moreover, Chaudhuri, Rao, Riesenfeld and Talwar present in [3] a quasi-polynomial-time sequential algorithm to find a minimum cost tree of maximum degree at most $\Delta^* + \mathcal{O}\left(\frac{\log n}{\log \log n}\right)$ and they conjecture that it is possible to obtain a polynomial-time sequential algorithm to find a minimum cost tree of maximum degree at most $\Delta^* + 1$.

In the paper, we propose a *distributed version* of Fischer's approximation algorithm that computes a MWST of maximum 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. Our distributed algorithm requires $\mathcal{O}\left(n^{3+1/\ln b}\right)$ messages, $\mathcal{O}\left(\Delta n^{2+1/\ln b}\right)$ (virtual) time, where Δ is the maximum degree of the initial minimum weight spanning tree and only $\mathcal{O}(n)$ space per node. From the complexity analysis of our distributed algorithm (see the proof of Lemma 2), one can derive that Fischer's sequential algorithm can actually be performed in $\mathcal{O}\left(n^{3+1/\ln b}\right)$ time, which improves on the upper bound on the time complexity given in Fischer's paper [4] (see Corollary 1).
To our knowledge, this is the first distributed approximation algorithm for the minimum degree minimum weight spanning tree problem.

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

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, though each node only knows its own incident edges, no node has a global knowledge of the network topology (e.g., no node is aware 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 (locally) 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 [9]). The *message complexity* of a distributed algorithm is the total number of messages sent over the edges (and received). We also assume that each message contains $\mathcal{O}(\log n + R)$ bits, where $n = |V|$ 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.

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* (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.
Let $N_G(u)$ and $N_T(u)$ denote the set of neighbors of node u in G and in T, respectively. In order to simplify the notation, we also denote $xy \in G$ any edge $xy \in E(G)$ and $v \in G$ any node $v \in V(G)$, respectively.
Let T be a tree on n nodes. The **rank** of T is defined as the ordered n-tuple $(t_n, \ldots, t_1)$ where t_i denotes the number of nodes of degree i in T. Also define a lexicographical order on these ranks; a tree T' on n nodes is of lower rank that T *iff* $t'_j < t_j$ 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 two edges; a swap is said *cost-neutral* if the edges exchanged have equal weights. Consider a swap between the edges $xw \in T$ and $uv \notin T$, with $x \notin \{u, v\}$. The 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 [4]) 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.*

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, as can be deduced from the proof of Theorem 1 in Fischer's paper [4] (see the Appendix). Fischer's sequential 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 edge vw 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 [4] that each iteration of his previous sequential algorithm takes $\mathcal{O}\left(n^2\right)$ time and that the number of iterations can be bounded by $\mathcal{O}\left(n^{2+1/\ln b}\right)$ (see the Appendix). Therefore, Fischer's sequential algorithm computes a locally optimal minimum weight spanning tree in time $\mathcal{O}(n^{4+1/\ln b})$.

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 FIFO 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. (Recall that MWST distributed algorithms are $\Theta(|E| + n \log n)$ message optimal, see e.g. [1, 6, 9], while the best time complexity achieved is $\mathcal{O}(n)$ 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.

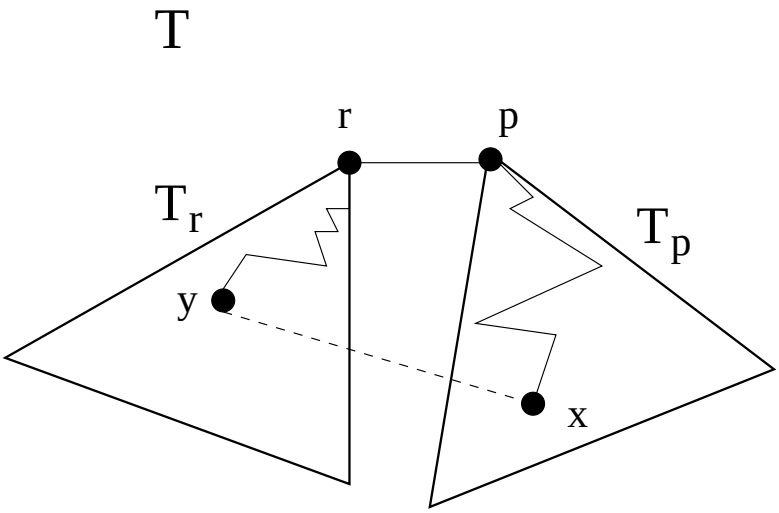

FIGURE 1: Example of a cost-neutral swap between edges $pr \in T$ and $xy \notin T$.

• **Initialization phase.** Let Δ_T be the maximum degree of the current 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 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 involving the edge pr, it needs to find a neighbor y in G such that $y \in T_r$ and $xy \notin T$. Similarly, node r initiates a broadcast by forwarding the token $\langle rp \rangle$ to all nodes in T_r.

• **Search-edge phase.** Whenever an edge pr in the current MWST T ends the Initialization phase (i.e., all nodes in T_p have received the token $\langle pr \rangle$ and all nodes in T_r have received the token $\langle rp \rangle$), 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 $y \in T_r$ (i.e., $xy \notin T$), for which $w(pr) = w(xy)$ and such that the maximal value of the degrees of p and r in T is at least greater than the degree of x and y plus 2. 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, it is possible that others nodes in T, distinct from p, have also reached this phase. So, all nodes reaching this phase (i.e. the possible candidates) start an election procedure by forwarding their identities on T. From all the possible candidates, 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: (T denotes 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 in T 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 node $r \in T$, the variable $Father_p[r]$ denotes the neighbor of p in T closer to node r. *Default value:* $Father_p[r] = p$, for each node $r \in T$.

- $Side_p[uv]$: *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 variable $CountSide_p[uv]$ maintains the number of neighbors of p which already instanced the variable $Side_p[uv]$. 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-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-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 winner node during an Edge-election phase. *Default value:* $NodeElec_p$ = *udef.*

- $EdgeElec_p$: *List of* 5 *integers.* The variable maintains the pair of elected edges (if any) in view of an appropriate swap 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 $d \stackrel{def}{=} \max\{d_p^T, d_p[r]\}$. *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 and 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 candidates) 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. Since all vertices are candidates in the election, 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 $\langle \textbf{side}, p, z, p\rangle$ to each node $r \in N_T(p)$ s.t. $r \neq z$;

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

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

It is well known that a distributed election procedure on a tree on n nodes requires $\mathcal{O}(n)$ messages and time. So, Step 1. of the Initialization phase is performed in $\mathcal{O}(n)$ time and uses $\mathcal{O}(n)$ messages. During Step 2.1. of the previous phase, for each node

p and each neighbor node z of p, the node p sends a message $\langle \textbf{side}, p, z, p\rangle$ to each one of its neighboring nodes $r \neq z$ in order to inform that the side of node r w.r.t. the edge pz is p. Moreover, the last parameter p serves node r to know from which node it received such a message. Notice that each edge $e \in T$ is traversed by $n-2$ messages of type **side** coming from all edges of T but e. So, the number of messages of type **side** in T is $(n-1)(n-2)$. If a leaf p receives a message $\langle \textbf{side}, u, v, q\rangle$ from node q, it sends a message $\langle \textbf{end-side}, u, v\rangle$ to q in order to inform node u that he has instanced the variable $Side_p[uv]$. The number of messages of type **end-side** is also $(n-1)(n-2)$. Moreover, if a node p receives a message $\langle \textbf{end-side}, p, v\rangle$ and any node k in the subtrees rooted at each of its neighbors but v have instanced their variable $Side_k[pv]$ (i.e. $CountSide_p[pv] = |N_T(p)| - 1$), it sends a message $\langle \textbf{check-side}, p\rangle$ to node v in order to inform node v that he has terminated the Initialization phase w.r.t. the edge pv and waits for node v to terminate such a phase as well, before edge pv begins the **Search-edge phase**. So, the number of messages in the Initialization phase is $\mathcal{O}\left(n^2\right)$ and it is easily deduced that the time complexity is $\mathcal{O}(\Delta D)$, where Δ is the maximum degree of the current tree and D is its diameter, respectively.

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.
   Before node p begins this phase, it waits that for all node q in $N_T(p)$ the condition $EndInit_p[q] = 2$ holds. Let W_p be a subset of $N_T(p)$ s.t. for all $q \in W_p$ we have that $\max\{d_p^T, d_p[q]\} \geq \Delta_T - \lceil \log_b n \rceil$.

   3.1. **If** $W_p \neq \emptyset$ **then**
   For each node $q \in W_p$ **do**
   send $\langle \textbf{change}, p, q, \mu_{pq}, w_p[q]\rangle$ to each node $r \in N_T(p)$, with $r \neq q$, and where $\mu_{pq} \stackrel{def}{=} \max\{d_p^T, d_p[q]\}$;
   else
   $\#Fails_p \leftarrow \#Fails_p + 1$;
   send $\langle \textbf{end}, p, p\rangle$ to each node $r \in N_T(p)$.

   3.2. Upon receipt of $\langle \textbf{change}, u, v, \mu_{uv}, w_u[v]\rangle$

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

3.3. Upon receipt of $\langle \textbf{find}, u, v, q, r \rangle$

- $EdgeFind_p[uv] \leftarrow 1$;
- **If** $p \neq u$ **then** send $\langle \textbf{find}, u, v, q, r \rangle$ 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 $\langle \textbf{fail}, u, v \rangle$

- $CountFail_p[uv] \leftarrow CountFail_p[uv] + 1$;
- **If** $EdgeFind_p[uv] \neq 1$ **then**
  If $p \neq u$ **and** $CountFail_p[uv] = |N_T(p)| - 1$ **then** send $\langle \textbf{fail}, u, v \rangle$ to node $Father_p[u]$;
  If $p = u$ **and** $CountFail_p[uv] = |N_T(p)| - 1$ **then**
  $NeighFail_p \leftarrow NeighFail_p + 1$;
  If $NeighFail_p = |W_p|$ **then** $\#Fails_p \leftarrow \#Fails_p + 1$ and send $\langle \textbf{end}, p, p \rangle$ to each node $r \in N_T(p)$ in order to check whether the algorithm is terminated;

3.5. Upon receipt of $\langle \textbf{end}, r, q \rangle$

- **If** $\#Fails_p < |T|$ **then** $\#Fails_p \leftarrow \#Fails_p + 1$;
- send $\langle \textbf{end}, r, p \rangle$ to each node $z \neq q$ in $N_T(p)$;
- **If** $\#Fails_p = |T|$ **then** the algorithm terminates for p;

Let $k \leq n - 1$ be the number of edges pq in T such that the sets W_p and W_q (defined in Step 3) are not empty during the Search-edge phase. As in the analysis of the Initialization phase, it is easy to conclude that the number of messages $\langle \textbf{change} \rangle$ in T is at most $k(n-2)$ and so, the number of such messages is at most $(n-1)(n-2)$. If a node p receives a message $\langle \textbf{change}, u, v, d, w \rangle$ and finds an edge pr in G but not in T for an appropriate improvement, it sends a message $\langle \textbf{find}, u, v, p, r \rangle$ to node $Father_p[u]$ to inform node u (i.e. edge uv) of success. The number of messages $\langle \textbf{find} \rangle$ is also at most $(n-1)(n-2)$. If a node $p \neq u$ receives a message $\langle \textbf{fail}, u, v \rangle$ from any node k of the subtrees rooted at each of its neighbors but node $Father_p[uv]$ (i.e. $CountFail_p[uv] = |N_T(p)| - 1$), it informs node $Father_p[uv]$ of an impossibility to find an edge for an appropriate improvement with edge uv. On the other hand, if $p = u$ and $CountFail_p[uv] = |N_T(p)| - 1$ during the reception of a $\langle \textbf{fail}, u, v \rangle$ message, it means that the edge pv, with $v \in W_p$, is unable to realize an appropriate edge-swap, and thus the variable $NeighFail_p[uv]$ is incremented by one unity. Now, if all the edges pq for all nodes $q \in W_p$ are unable to realize an appropriate edge-swap in T (i.e. $NeighFail_p = |W_p|$), then the variable $\#Fails_p$ is incremented by one unity and p broadcasts a message $\langle \textbf{end}, p, p \rangle$ to all other nodes in the tree, in order to check whether the algorithm is terminated (the second parameter p in the message is to inform that the message comes from p).
Notice that in the case when p has an empty set W_p, the node p increments its variable $\#Fails_p$ by one and also broadcasts the message $\langle \textbf{end}, p, p \rangle$ to all other nodes in the

tree (see Step 3.1). Since the number of messages $\langle$**fail**$\rangle$ and $\langle$**end**$\rangle$ can be bounded by $\mathcal{O}\left(n^2\right)$, the total number of messages in such a phase is bounded by $\mathcal{O}\left(n^2\right)$. The time required in such a phase is also bounded, as in the previous phase, by $\mathcal{O}(\Delta D)$, where Δ is the maximum degree of the current tree and D is its diameter, respectively.

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 changed for an unused edge $xy \in G$. However, it is possible that other nodes in T, distinct from p, also reached this phase. So, all nodes reaching this phase (i.e. the possible candidates) start an election procedure by forwarding their identities on T. From all possible candidates, the elected node is the one with the minimal identity. 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**$\rangle$ and $\langle$**end**$\rangle$ received by p is ignored. Now, let $\mu_{pr} \stackrel{def}{=} \max\left\{d_p^T, d_p[r]\right\}$.

   4.1. **If** $Mode_p = non\text{-}election$ **then**

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

   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 $\langle \mathbf{lost}, q, p\rangle$ to each node $r \in N_T(p)$ s.t. $r \neq t$.

The previous Edge-election phase is a classical distributed election in a tree of n nodes and requires $\mathcal{O}(n)$ messages and time.

Edge-exchange phase:

5. When node p wins the Edge-election phase, i.e. $State_p = winner$, with $EdgeElec_p = (\mu_{pr}, p, r, x, y)$, the edge $pr \in T$ is ready to be changed 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 $\langle \mathbf{disconnect}\rangle$, $\langle \mathbf{connect}\rangle$, $\langle \mathbf{new}\rangle$, and $\langle \mathbf{nround}\rangle$ is ignored. So, during this phase the following is done.

   5.1. • Send $\langle \mathbf{disconnect}, r, p\rangle$ 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 $\langle \mathbf{connect}, p, r, x, y\rangle$ to each node $t \in N_T(p) : t \neq r$;

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

   5.3. Upon receipt of $\langle \mathbf{connect}, u, v, x, y\rangle$
   • **If** $p \neq x$ and p is not a leaf **then** send $\langle \mathbf{connect}, u, v, x, y\rangle$ to each node $t \in N_T(p) : t \neq Father_p[u]$;
   • **If** $p = x$ **then**
   send $\langle \mathbf{new}, y, p, u, v\rangle$ 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 $\langle \mathbf{new}, p, q, u, v\rangle$
   • $Neigh_p[q] \leftarrow branch$; $d_p[q] \leftarrow d_p[q] + 1$; $d_p^T \leftarrow d_p^T + 1$;
   • Actualize the data structure of T, i.e. $T \leftarrow (T \setminus \{uv\}) \cup \{pq\}$;
   • send $\langle \mathbf{nround}, p, p, q, u, v\rangle$ to each node $t \in N_T(p)$;

   5.5. Upon receipt of $\langle \mathbf{nround}, z, u, v, x, y\rangle$
   • Actualize the data structure of T, i.e. $T \leftarrow (T \setminus \{uv\}) \cup \{xy\}$;
   • **If** p is not a leaf **then** send $\langle \mathbf{nround}, p, u, v, x, y\rangle$ 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 */

During this last phase, the elected node p begins the search for the node $x \in T_p$ in order to realize the appropriate edge-swap between the edges pr and xy. So, p broadcasts the message $\langle \mathbf{connect}, p, r, x, y\rangle$ to the nodes in the subtree T_p w.r.t. the edge pr. When node x is found, it sends a message $\langle \mathbf{new}, y, x, u, v\rangle$ to node y in order to inform that, from now on, the edge xy belongs to the current tree T. When node y receives a message $\langle \mathbf{new}, y, x, u, v\rangle$ from node x, it actualizes the data structure of the new current tree in its local memory: the edge uv is deleted and the edge xy is added.
For example, if the tree is represented by an adjacent list, then $\mathcal{O}(n)$ local space and time

are required to actualize the new current tree. Moreover, node y broadcasts the message $\langle \mathbf{nround}, y, y, x, u, v \rangle$ to all its neighbors in the tree, where the first parameter denotes the sender of the message, and the remaining four parameters are used to actualize the data structure representing the current tree. Since only the nodes p and y broadcast an information to all other nodes in the tree, such a phase takes $\mathcal{O}(n)$ messages and time.

4 Correctness and Complexity

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

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. In order to prove that such a condition is always reached by the algorithm, consider the Search-edge phase. For a fixed node p, let W_p be the subset of neighbors of p in T such that for all $q \in W_p$, the edge pq is ready for a possible appropriate edge-swap. In Step 3.1. of the mentioned phase, if W_p is empty, the node p increments its variable $\#Fails_p$ by one unity and broadcasts the message $\langle \mathbf{end} \rangle$ to all other nodes in the tree. Otherwise, if W_p is not empty but all edges pq for all nodes $q \in W_p$ are unable to realize an appropriate edge-swap in T (i.e. $NeighFail_p = |W_p|$), then p increment its variable $\#Fails_p$ by one unity and broadcast a message $\langle \mathbf{end} \rangle$ to all other nodes in the tree (see Step 3.4 of the Search-edge phase). Moreover, each time p receives an $\langle \mathbf{end} \rangle$ message, it increments its variable $\#Fails_p$ by one unity (see Step 3.5 of the Search-edge phase). So, if no edge in T is able to realize an appropriate edge-swap, it means that each node p in T has received $|T| - 1$ $\langle \mathbf{end} \rangle$ messages, and thus, $\#Fails_p = |T|$. Hence, by definition, T is indeed a locally optimal minimum weight spanning tree and the algorithm is correct. □

The following lemma is easily deduced from the previous comments, after, each phase of the distributed algorithm.

Lemma 1 *Each round of the distributed algorithm in Section 3 requires $\mathcal{O}\left(n^2\right)$ messages and $\mathcal{O}(\Delta n)$ time, where Δ is the maximum degree of the initial MWST.*

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 $\mathcal{O}\left(n^{1+1/\ln b}\right)$.*

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 [8].
Let Δ_T be the maximum degree of the current MWST T on n nodes during a round of the algorithm. Let $\delta \stackrel{def}{=} \max\{\Delta_T - \lceil \log_b n \rceil, 0\}$ 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. two rounds), Φ_T decreases of at least one unit. Hence, the algorithm finds a locally optimal MWST in $\mathcal{O}\left(n^{1+1/\ln b}\right)$ rounds. □

Theorem 3 *For any constant $b > 1$, the distributed algorithm in Section 3 requires $\mathcal{O}\left(n^{3+1/\ln b}\right)$ messages and $\mathcal{O}(n)$ 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. Moreover, the algorithm takes $\mathcal{O}\left(\Delta n^{2+1/\ln b}\right)$ (virtual) time, where Δ is the maximum degree of an initial MWST.*

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 specifically maintains the variables $Side_p[uv]$, $CountSide_p[uv]$, $CountFail_p[uv]$ and $EdgeFind_p[uv]$ (see Section 3.2). Since T is a tree of n nodes, its number of edges is $n - 1$, and so the algorithm requires $\mathcal{O}(n)$ space per node. Finally, by Theorems 1 and 2, and Lemmas 1 and 2, the proof 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 [4] (Section 2) finds a minimum weight spanning tree of maximum degree at most $b\Delta^* + \lceil \log_b n \rceil$ in $\mathcal{O}\left(n^{3+1/\ln b}\right)$ time.*

5 Concluding Remarks

In the paper, we present a distributed approximation algorithm which computes a minimum weight spanning tree of maximum degree at most $b\Delta^* + \lceil \log_b n \rceil$, for any constant

$b > 1$. The message complexity of the algorithm is $\mathcal{O}(n^{3+1/\ln b})$, its time complexity is $\mathcal{O}\left(\Delta n^{2+1/\ln b}\right)$, where Δ is the maximum degree of the initial MWST of G and $\mathcal{O}(n)$ space per node is required.
Note that since the proof of Lemma 2 works also in the sequential case, the time complexity obtained actually improves on Fisher's upper bound in [4].

To our knowledge, this is the first distributed approximation algorithm for the minimum degree minimum weight spanning tree problem.

Acknowledgments

We would like to thank an anonymous referee for providing helpful comments and pointing out serious flaws in the original paper.

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] K. Chaudhuri, S. Rao, S. Riesenfeld, K. Talwar. What would Edmonds do ? Augmenting paths and witnesses for degree-bounded MSTs, *In Proc. of Approx-Random*, LNCS 3624, 26-39, 2005.

[4] T. Fischer. Optimizing the degree of minimum weight spanning trees, *Technical Report 93-1338*, Department of Computer Science, Cornell University, Ithaca, NY, USA, 1993.

[5] 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).

[6] 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).

[7] M. R. Garey, D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman Eds., San Francisco, 1979.

[8] 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.

[9] G. Tel. *Introduction to Distributed Algorithms*, Cambridge University Press, 1994.

Appendix

In order to facilitate the reading of the paper, the proof of Fischer's Theorem 1 in Section 2 and Fischer's proof concerning the number of iterations needed for his sequential algorithm are both given.

Proof of Theorem 1 (Fischer [4]). 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 ξ in the range $\Delta_T - \lceil \log_b n \rceil \leq \xi \leq \Delta_T$ such that $|S_{\xi-1}|/|S_\xi| \leq b$. Suppose we choose an integer ξ to satisfy this property, and remove from T the edges adjacent to nodes in S_ξ. Let T_ξ denote the remaining edges of T. As T is initially connected, then there are at least $\xi|S_\xi| + 1 - (|S_\xi| - 1)$ or $(\xi - 1)|S_\xi| + 2$ connected components in T_ξ.

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

Consider an edge $vw \in G$ not in T between two components of T_ξ. Let P^T denote the vw-path in T, and P^T_ξ denote those edges of P^T which appear in G_ξ, the edges on the path which are adjacent to any node in S_ξ. Suppose neither v nor w is in $S_{\xi-1}$. 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_ξ. Since vw and P^T_ξ form a cycle in G_ξ, this implies that vw may not participate in a MWST of G_ξ. Therefore, only edges which are adjacent to nodes in $S_{\xi-1}$ may participate in a MWST of G_ξ, and any MWST of G must contain at least $(\xi - 1)|S_\xi| + 1$ edges that are adjacent to $S_{\xi-1}$.

Above, we choose ξ to satisfy the inequality $|S_{\xi-1}|/b \leq |S_\xi|$. Substituting, we see there must be at least $\dfrac{(\xi - 1)|S_{\xi-1}|}{b+1}$ edges adjacent to nodes in $S_{\xi-1}$. Therefore, the average degree of a node in $S_{\xi-1}$ must be at least

$$\frac{(\xi - 1)|S_{\xi-1}| + b}{b|S_{\xi-1}|},$$

and thus, $\Delta^* > \frac{\xi-1}{b}$.
Combining this with the possible range for ξ yields $\Delta_T \leq b\Delta^* + \lceil \log_b n \rceil$. □

Regarding the number of iterations required for Fischer's sequential algorithm, the following lemma is proved in [4].

Lemma 3 *(Lemma 2.6 in [4]) Fischer's sequential algorithm presented in Section 2 will terminate in $\mathcal{O}(n^{2+1/\ln b})$ iterations.*

Proof: Define the potential of a vertex v to be e^{d_v}, where e is the Euler's constant and d_v is the degree of v in the current tree. Define $\Phi = \sum_{v \in V} e^{d_v}$. Let Δ be the maximum

degree value of the current tree and let $i = \Delta - \lceil \log_b n \rceil$, the lower limit on the degree for our improvements. Since we only perform swaps which only improve the degree of some vertex in S_i (see definition in the proof of previous theorem in this appendix), the reduction in Φ resulting from a swap is at least

$$e^i + 2e^{i-2} - 3e^{i-1} = (e-1)(e-2)e^{i-2} \geq ce^i \quad \text{for some constant } c.$$

Now $\Phi \leq ne^\Delta$, so each swap reduces Φ by at least a fraction of

$$\frac{ce^i}{ne^\Delta} = \frac{ce^{-\lceil \log_b n \rceil}}{n} \geq \frac{c'}{n^{1+1/\log_b e}} \quad \text{for some constant } c'.$$

Therefore, in $\mathcal{O}\left(n^{1+1/\ln b}\right)$ iterations, the potential reduces by a constant fraction, and after $\mathcal{O}\left(n^{2+1/\ln b}\right)$ iterations, the algorithm must halt. □