A Distributed Algorithm for the Minimum Diameter Spanning Tree Problem

Marc Bui, Franck Butelle, Christian Lavault

A Distributed Algorithm for Constructing a Minimum Diameter Spanning Tree

Marc Bui

Laboratoire de Recherche LDCI, Université Paris 8, France

Franck Butelle *

Laboratoire de Recherche LIPN – CNRS UMR 7030, Université Paris 13, France

Christian Lavault

Laboratoire de Recherche LIPN – CNRS UMR 7030, Université Paris 13, France

Abstract

We present a new algorithm, which solves the problem of distributively finding a minimum diameter spanning tree of any (non-negatively) real-weighted graph $G = (V, E, \omega)$. As an intermediate step, we use a new, fast, linear-time all-pairs shortest paths distributed algorithm to find an absolute center of G. The resulting distributed algorithm is asynchronous, it works for named asynchronous arbitrary networks and achieves $O(|V|)$ time complexity and $O(|V|\,|E|)$ message complexity.

Key words: Spanning trees, Minimum diameter spanning trees, Shortest paths, Shortest paths trees, All-pairs shortest paths, Absolute centers.

1 Introduction

Many computer communication networks require nodes to broadcast information to other nodes for network control purposes; this is done effi ciently by sending messages over a spanning tree of the network. Now, optimizing the worst-case message propagation over a spanning tree is naturally achieved by reducing the diameter to a minimum.

* Contact author: butelle@lipn.univ-paris13.fr

The use of a control structure spanning the entire network is a fundamental issue in distributed systems and interconnection networks. Given a network, a distributed algorithm is said to be *total* iff all nodes participate in the computation. Now, all total distributed algorithms have a time complexity of $\Omega(D)$, where D is the network diameter (either in terms of hops, or according to the wider sense given by Christophides in [9]). Therefore, having a spanning tree with minimum diameter of arbitrary networks makes it possible to design a wide variety of time-efficient distributed algorithms. In order to construct such a spanning tree, all-pairs shortest paths in the graph are needed first. Several distributed algorithms already solve the problem on various assumptions. However, our requirements are more general than the usual ones. For example, we design a "process terminating" algorithm for (weighted) networks with no common knowledge shared between the processes. (See assumptions in Subsection 1.2 below.)

1.1 Model, Notations and Definitions

A distributed system is a standard point-to-point asynchronous network consisting of n communicating processes connected by m bidirectional channels. Each process has a local non-shared memory and can communicate by sending messages to and receiving messages from its neighbours. A single process can transmit to and receive from more than one neighbour at a time.

The network topology is described by a finite, weighted, connected and undirected graph $G = (V, E, \omega)$, devoid of multiple edges and loop-free. G is a structure which consists of a finite set of nodes V and a finite set of edges E with real-valued weights; each edge e is incident to the elements of an unordered pair of nodes (u, v). In the distributed system, V represents the processes, while E represents the (weighted) bidirectional communication channels operating between neighbouring processes [19]. We assume that, for all $(u, v) \in E$, $\omega(u, v) = \omega(v, u)$ and, to shorten the notation, $\omega(u, v) = \omega(e)$ denotes the real-valued weight of edge $e = (u, v)$. (Assumptions on real-valued weights of edges are specified in the next two Subsections 1.2 and 1.3.) Throughout, we let $|V| = n$, $|E| = m$ and, according to the context, we use G to represent the network or the weighted graph, indistinctly.

The weight of a path $[u_0, \ldots, u_k]$ of G ($u_i \in V$, $0 \le i \le k$) is defined as $\sum_{0 \le i \le k-1} \omega(u_i, u_{i+1})$. For all nodes u and v in V, the *distance* from u to v, denoted $d(u, v) = d_G(u, v) = d(v, u) = d_G(v, u)$, is the lowest weight of any path length from u to v in G. The largest (minimal) distance from a node v to all other nodes in V, denoted $ecc(v) = ecc_G(v)$, is the *eccentricity* of node v: $ecc(v) = \max_{u \in V} d(u, v)$ [9]. An *absolute center* of G is defined as a node (not necessarily unique) achieving the smallest eccentricity in G.

$D = D(G)$ denotes the *diameter* of G, defined as $D = \max_{v \in V} ecc(v)$ (see [9]), and

$R = R(G)$ the *radius* of G, is defined as $R = \min_{v \in V} ecc(v)$. Finally, $\Psi(u) = \Psi_G(u)$ represents the *shortest paths tree* (SPT) of G rooted at node u: $(\forall v \in V)\ d_{\Psi(u)}(u,v) = d(u,v)$. $\Psi(u)$ is chosen *uniquely* among all the shortest paths trees of G rooted at node u; whenever there is a tie between any two length paths $d(u,v)$, it is broken by choosing the path with a second node of minimal identity. The set of all SPTs of G is denoted $\Psi = \Psi(G)$. When it is clear from the context, the name of the graph is omitted.

In the remainder of the paper, we denote problems as "the (MDST) problem", "the (MST) problem", "the (GMDST) problem", etc. (see definitions in Subsection 1.4). Distributed algorithms are denoted in italics, e.g. "algorithm *MDST*". Finally, "MDST", "APSPs" and "SPT" abbreviate "minimum diameter spanning tree", "all-pairs shortest paths" and "shortest paths tree", respectively.

1.2 The Problem

Given a weighted graph $G = (V, E, \omega)$, the **(MDST) problem** is to find a spanning tree of G of minimum diameter D (according to the definition of D).

Note that the (MDST) problem assumes G to be a non-negatively real-weighted graph (i.e., $\forall e \in E\ \ \omega(e) \in \mathbb{R}^+$). Indeed, the (MDST) problem is known to be NP-hard if we allow negative length cycles in G (cf. Camerini *et al.* [7]).

In spite of the fact that the (MDST) problem requires arbitrary non-negative real-valued edges weights, our distributed MDST algorithm is process terminating (i.e., a proper distributed termination is completed [19]). This is generally not the case on the above requirement. When weights are assumed to be real-valued, a common (additional) knowledge of a bound on the size of the network is usually necessary for APSPs algorithms to process terminate (see e.g. [1,4,18]). By contrast, no "structural information" is assumed in our algorithm, neither topological (e.g., size or bound on the size of the network), nor a sense of direction, etc. (see Subsection 2.2).

1.3 Assumptions

In addition to the above general hypothesis of the (MDST) problem, we need the following assumptions on the network.

- Processes are faultless, and the communication channels are faithful, lossless and order-preserving (FIFO).
- All processes have *distinct* identities (*ID*s). (G is called a "named network", by contrast with "anonymous networks".) We need distinct *ID*s to compute the AP-

SPs routing tables of G at each process of the network. For the sake of simplicity, *ID*s are also assumed to be non-negative distinct integers.

Each process must distinguish between its ports, but has no *a priori* knowledge of its neighbours *ID*s. Actually, any process knows the *ID* of a sending process after reception of its first message. Therefore, we assume w.l.o.g. (and up to $n-1$ messages at most) that a process knows the *ID* of each of its neighbours from scratch (see protocol *APSP* in Subsection 2.1.2).

- Of course, each node also knows the weights of its adjacent edges. However, edges weights do not satisfy the triangular inequality.
- Let $\mathcal{A}$ be a distributed algorithm defined on G. A non-empty subset of nodes of V, called *initiators*, simultaneously start algorithm $\mathcal{A}$. In other words, an external event (such as a user request, for example), impels the initiators to trigger the algorithm. Other (non-initiating) nodes "wake up" upon receipt of a first message.
- In a reliable asynchronous network, we measure the communication complexity of an algorithm $\mathcal{A}$ in terms of the maximal number of messages that are received, during any execution of $\mathcal{A}$. We also take into account the number of bits in the messages (or message size) : this yields the "bit complexity" of $\mathcal{A}$. For measuring the time complexity of $\mathcal{A}$, we use the definition of standard time complexity given in [19,20]. Standard time complexity is defined on *"Asynchronous Bounded Delay networks"* (ABD networks): we assume an upper bound transmission delay time of τ for *each* message in a channel; τ is then the "standard time unit" in G.

1.4 Related Works and Results

The small amount of literature related to the (MDST) problem mostly deals either with graph problems in the Euclidian plane (geometric minimum diameter spanning tree: the (GMDST) problem), or with the Steiner spanning tree construction (see [14,15]). The (MDST) problem is clearly a generalization of the (GMDST) problem. The sequential problem has been addressed by some authors (see for example [9]).

Surprisingly, despite the importance of having an MDST in arbitrary distributed systems, only few papers have addressed the question of how to design algorithms which construct such spanning trees. Finding and maintaining a *minimum spanning tree* (the (MST) problem) has been extensively studied in the literature (e.g. [2,3,10,12]). More recently, the problem of maintaining a *small* diameter was however solved in [16], and the distributed (MDST) problem was addressed in [5].

1.5 *Main contributions of the paper*

Our algorithm *APSP* is a generalization of APSP algorithms on graphs with unit weights (weights with value 1) to the case of non-negatively real-weighted graphs. To our knowledge, our MDST finding algorithm is also the first which *distributively* solves the (MDST) problem [5]. The algorithm *MDST* works for named arbitrary network topologies with asynchronous communications. It achieves an "efficient" $O(n)$ time complexity and $O(nm(\log n + \log W))$ bits communication complexity, where W is the largest weight of a channel. (An $O(n)$ time complexity may be considered "efficient", though not optimal, since the construction of a spanning tree costs at least $\Omega(D)$ in time).

The paper is organized as follows. In Section 2 we present a high-level description of the protocol *APSP*, a formal design of the procedure *Gamma_star* and the algorithm *MDST*. Section 3 is devoted to proofs and complexity analysis of the algorithm. Finally, concluding remarks are given in Section 4.

2 The Algorithm

2.1 *A High-Level Description*

2.1.1 *Main Issues*

First, we recall in Lemma 1 that the (MDST) problem for a weighted graph G is (polynomially) reducible to the absolute center problem for G. Then, we constructively find and compute an absolute center of G by using its APSPs routing tables in Lemma 2.

In summary, given a positively weighted graph G, the main steps of our algorithm for the (MDST) problem are the following:

(1) The computation of APSPs in G;
(2) The computation of an absolute center of G (procedure *Gamma_star*(e) in Subsection 2.2);
(3) The construction of an MDST of G, and the transmission of the knowledge of that MDST to each node within the network G.

2.1.2 *Construction of an MDST*

The definition of the eccentricity is generalized as follows. We view an edge (u,v) with weight ω as a continuous interval of length ω, and for any $0 < \alpha < \omega$ we allow an insertion of a "dummy node" γ and replace the edge (u,v) by a pair of edges: (u,γ) with weight α and (γ,v) with weight $\omega-\alpha$.

According to the definition, the eccentricity $ecc(\gamma)$ of a *general node* γ (i.e., either an actual node of V, or a dummy node) is clearly given by $ecc(\gamma)=\max_{z\in V} d(\gamma,z)$. A node γ^* such that $ecc(\gamma^*)=\min_\gamma ecc(\gamma)$ is called *an absolute center* of the graph. Recall that γ^* always exists in a connected graph and that it is not unique in general. Moreover, an absolute center of G is usually one of the dummy nodes.

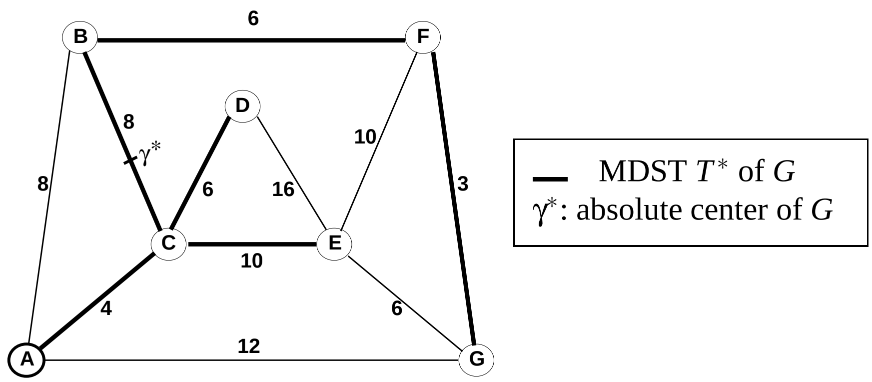

Fig. 1. Example of an MDST T^* ($D(G)=22$ and $D(T^*)=27$). T^* is neither a shortest paths tree, nor a minimum spanning tree of G.

Similarly, the definition of $\Psi(u)$ is also generalized to account for the dummy nodes. Finding an MDST actually reduces to searching for an absolute center γ^* of G: the SPT rooted at γ^* is an MDST of G. Such is the purpose of the following Lemma 1.

Lemma 1 [7] *Given a weighted graph G, the (MDST) problem for G is (polynomially) reducible to the problem of finding an absolute center of G.*

2.1.3 Computation of an absolute center of a graph

The idea of computing absolute p-centers was first introduced by Hakimi, see for example [13]. Here we address the computation of an absolute 1-center. According to the results in [9], we need the following lemma (called Hakimi's method) to find an absolute center of G.

Lemma 2 *Let $G=(V,E,\omega)$ be a weighted graph. For each edge $e\in E$, let γ_e be the set of all the general nodes of G which achieve a minimal eccentricity for e. A node achieving the minimal eccentricity among all nodes in $\bigcup_{e\in E}\gamma_e$ is an absolute center. Finding a minimum absolute center of G is thus achieved in polynomial time.*

PROOF. (the proof is constructive) *(i)* For each edge $e=(u,v)$, let $\alpha = d(u,\gamma)$. Since the distance $d(\gamma,z)$ is the length of a path $[\gamma,u,\ldots,z]$ or a path $[\gamma,v,\ldots,z]$,

$$ecc(\gamma) = \max_{z\in V} d(\gamma,z) = \max_{z\in V} \min(\alpha + d(u,z), \omega(u,v) - \alpha + d(v,z)). \tag{1}$$

If we plot $f_z^+(\alpha) = \alpha + d(u,z)$ and $f_z^-(\alpha) = -\alpha + \omega(u,v) + d(v,z)$ in Cartesian coordinates for fixed $z = z_0$, the real-valued functions $f_{z_0}^+(\alpha)$ and $f_{z_0}^-(\alpha)$ (separately depending on α in the range $[0,\omega(e)]$) are represented by two line segments $(S_1)_{z_0}$ and $(S_{-1})_{z_0}$, with slope $+1$ and -1, respectively. For a given $z = z_0$, the smallest of the two terms $f_{z_0}^+(\alpha)$ and $f_{z_0}^-(\alpha)$ in (1) define a piecewise linear function $f_{z_0}(\alpha)$ made of $(S_1)_{z_0}$ and $(S_{-1})_{z_0}$.

Let $B_e(\alpha)$ be the *upper boundary* ($\alpha \in [0,\omega(e)]$) of all the above $f_z(\alpha)$ ($\forall z \in V$). $B_e(\alpha)$ is a curve made up of piecewise linear segments, which passes through several local minima (see Figure 2). A point γ achieving the smallest minimal value (i.e. the global minimum) of $B_e(\alpha)$ is an absolute center γ_e^* of the edge e.

(ii) From the definition of γ_e^*, $\min_\gamma ecc(\gamma) = \min_{\gamma_e^*} s(\gamma_e^*)$; and γ^* achieves the minimal eccentricity. Therefore, an absolute center γ^* of the graph is found at any point where the minimum of all $ecc(\gamma_e^*)$s is attained. □

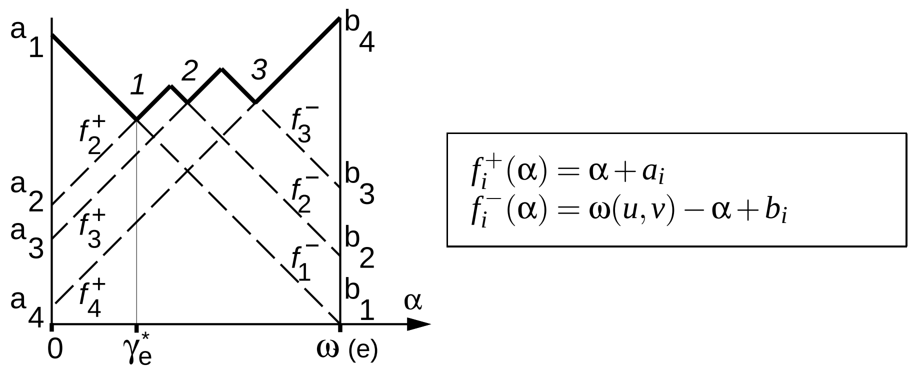

Fig. 2. Example of an upper boundary $B_e(\alpha)$

By Lemma 2, we may consider this method from an algorithmic viewpoint. For each $e=(u,v)\in E$, let

$$\mathcal{C}_e = \{(d_1,d_2) \mid (\forall z \in V) \quad d_1 = d(u,z) \text{ and } d_2 = d(v,z)\}.$$

Now, a pair (d_1', d_2') is said to *dominate* a pair (d_1,d_2) iff $d_1 \le d_1'$ and $d_2 \le d_2'$; namely, the function $f_{z'}(\alpha)$ defined by (d_1', d_2') is over $f_z(\alpha)$ defined by (d_1,d_2). Any such pair (d_1,d_2) will be ignored when it is dominated by another pair (d_1', d_2'). The local minima of the upper boundary $B_e(\alpha)$ (numbered from 1 to 3 in Figure 2) are located at the intersection of the segments $f_i^-(\alpha)$ and $f_{i+1}^+(\alpha)$, when all dominated pairs are removed. Sorting the set $\mathcal{C}_e$ in descending order, with respect to the first term of each remaining pair (d_1,d_2), yields the list $L_e = ((a_1,b_1),\ldots,(a_{|L_e|},b_{|L_e|}))$

consisting of all such ordered dominating pairs. Hence, the smallest minimum of $B_e(\alpha)$ for a given edge e clearly provides an absolute center γ_e^* (see the Procedure $Gamma\_star(e)$ in Subsection 2.2). By Lemma 2, once all the γ_e^*s are computed, we can obtain an absolute center γ^* of the graph G. Last, by Lemma 1, finding an MDST of G reduces to the problem of computing γ^*.

2.1.4 *All-Pairs Shortest Paths Algorithm (protocol* APSP*)*

In section 2.1.3, we consider the distances $d(u,z)$ and $d(v,z)$, for all $z \in V$ and for each edge $e = (u,v) \in E$. The latter distances must be computed by a distributed (process terminating) routing algorithm; the protocol *APSP* is designed for that purpose in Subsection 2.2.

2.1.5 *Construction and knowledge transmission of an MDST*

At the end of the protocol *APSP*, every node knows the node u_{min} with the smallest *ID* and a shortest path in G leading to u_{min}. Now, consider the collection of all paths $[u,\ldots,u_{min}]$ (computed by *APSP*), which start from a node $u \in V$ and end at node u_{min}. This collection forms a tree rooted at node u_{min} and, since it is an SPT of G, the information is exchanged "optimally" in the SPT $\Psi(u_{min})$ [1] . Hence, the number of messages needed to search an extremum in the tree $\Psi(u_{min})$ is at most $O(n)$ (with message size in $O(\log n + \log W)$).

When the computation of an absolute center γ^* of G is completed, the endpoint of γ^*'s edge having the smallest *ID* sends a message to u_{min} carrying the *ID* of γ^*. Upon receipt of the message, u_{min} forwards the information all over $\Psi(u_{min})$ (adding the same cost in time and messages).Therefore each node of G knows a route to γ^*, and the MDST is built as a common knowledge for all nodes.

2.2 *The Design of the Algorithm*

2.2.1 *Main Procedure*

The distributed algorithm *MDST* finds an MDST of an input weighted graph $G = (V,E,\omega)$ by computing an absolute center of G.

The algorithm is described from a node point of view. The algorithm assumes that each node u computes the following steps:

[1] In $\Psi(u_{min})$, the information is transmitted *"optimally"* in terms of time and messages, in the sense that each edge weight may be regarded as the message transmission delay of a channel.

- In step 1, u participates in the computation of the APSP. This computation gives the diameter and the radius of the graph G. Moreover it also gives u_{min}, the minimum node identity in the graph. (See section 2.1.5.)
- Steps 2 and 3 implement an adjacent edge selection procedure, by discarding heavy edges. The computation of the local minimum is accelerated with the help of an upper bound test. Note that the variable φ, used in the test, is a data structure with four fields: the best distance α from the first edge end, the upper bound value associed to α, the identities of the first and second edge ends. (Edge ends are ordered by increasing identities.)
- In steps 4, 5 and 6, node u participates in finding the minimum of all values φ.
- In step 7, the best φ is finally computed at the root of the tree $\Psi(u_{min})$ and next, it is broadcast to all nodes through $\Psi(u_{min})$.

For the sake of clarity, we use abstract record data types (with dot notation).

Algorithm *MDST* (for node u)

Type elt : **record**
$alpha\_best$, $upbound$: EdgeWeight ;
id_1, id_2 : NodeIdentity ;
end ;
Var φ, φ_u^* : elt ;
$Diam$, $Radius$, α, $localmin$: EdgeWeight ;
u_{min} : NodeIdentity ;
d_u : array of EdgeWeight ; *(* after step 1, $d_u[v] = d(u,v)$ *)*

(1) **For all** $v \in V$ Compute $d_u[v]$, $Diam$, $Radius$ and u_{min} ; *(* from protocol APSP *)*
(2) $\varphi.upbound \leftarrow Radius$;
(3) **While** $\varphi.upbound > Diam/2$ **do for** each edge $e = (u,v)$ s.t. $v > u$
  (a) $(\alpha, localmin) \leftarrow$ *Gamma_star*(e) ;
  (b) **If** $localmin < \varphi.upbound$ **then** $\varphi \leftarrow (\alpha, localmin, u, v)$;
(4) $\varphi_u^* \leftarrow \varphi$;
(5) **Wait for reception of** φ from each child of u in $\Psi(u_{min})$ **and do**
  if $\varphi_u^*.upbound < \varphi.upbound$ **then** $\varphi_u^* \leftarrow \varphi$;
(6) **Send** φ_u^* to parent in $\Psi(u_{min})$;
(7) **If** $u = u_{min}$ **then Send** φ_u^* to all its children
  else **Wait for reception of** φ^* from its parent **then Send** φ^* to all its children...

Now we describe the basic procedures used in the algorithm: first the protocol *APSP* and next the procedure *Gamma_star*(e).

2.2.2 The APSP algorithm

We need an algorithm that computes the all-pairs shortest paths in G and does process terminate without any structural information (e.g., the knowledge an upper bound on n). Our algorithm is based on the Netchange algorithm (see the proof in [18]), the Bellman-Ford algorithm (see [4]) and the α-synchroniser described in [1]. The three latter algorithms process terminate *iff* an upper bound on n is known. Otherwise, if the processes have no structural information, the above algorithms only "message terminate" (see [1,19]). However, proper distributed termination may be achieved without additional knowledge by using the same technique as designed in [8]. We now shortly describe the algorithm (from the viewpoint of node u, whose *ID* is id_u).

The protocol *APSP* is organized in phases after the first initialization step. This step starts initializing sets and variables (id_u is the selected *ID*): the distance to id_u is set to 0, while all others distances are set to ∞ and the set *Updatenodes* is initialized to $\emptyset$. Next, every phase of the algorithm consists of three steps:

- Step 1. Send to all neighbours the *ID* of the selected node and its distance to node u.
- Step 2. Wait for reception of the same number of messages sent in step 1 minus the number of inactive neighbours (see next paragraph). Upon receipt of a message, update distance tables. If the estimate of the distance to a node changes, add this node to the set *Updatenodes*. If an $\langle$**Inactive**$\rangle$ message is received from a neighbour, mark it *inactive*. When the awaited number of messages is received, start step 3.
- Step 3. Choose an active node from the set *Updatenodes* with the smallest distance to u and go to step 1. If no such node exists then send an $\langle$**Inactive**$\rangle$ messsage to each active neighbour; node u becomes an inactive node.

We need the following rules to make the algorithm process terminate.

(1) An inactive node forwards updating messages (if necessary) to its inactive neighbours.
(2) Only one $\langle$**Inactive**$\rangle$ message is sent from node u to a neighbour v and this message is the last message (of protocol *APSP*) from u to v.
(3) (from the previous rule) A node terminates only when two $\langle$**Inactive**$\rangle$ messages are received in each of its adjacent edges (one from each direction).

Thus, we designed a new distributed APSP protocol having a good message complexity, viz. $2mn$.

2.2.3 Procedure Gamma_star

Assume the list L_e (defined in section 2.1.3) to be already constructed (e.g. with a heap) whenever the routing tables are computed. The following procedure returns a value γ_e^* for any fixed edge $e \in E$.

Procedure *Gamma_star*(e)

> **var** min, α : real ; **Init** $min \leftarrow a_1$; $\alpha \leftarrow 0$;
> **For** $i \leftarrow 1$ to $|L_e| - 1$ **do**
> *compute the intersection* (x,y) *of segments* f_i^- *and* f_{i+1}^+ *:*
> $x \leftarrow \frac{1}{2}(\omega(e) - a_{i+1} + b_i)$;
> $y \leftarrow \frac{1}{2}(\omega(e) + a_{i+1} + b_i)$;
> **if** $y < min$ **then** $min \leftarrow y$; $\alpha \leftarrow x$;
> **Return**(α,min)

Remark 3 *Recall that for each edge $e = (u,v)$ of G with weight $\omega(e)$ and for any given $z \in V$, d_1 and d_2 are the distances $d_1 = d(u,z)$ and $d_2 = d(v,z)$. Moreover, all pairs (a_i,b_i) ($1 \le i \le |L_e|$) are those ordered pairs (d_1,d_2) of the list L_e which are dominating pairs (see the proof of Lemma 1).*

3 Analysis

For the purpose of the complexity analysis, let $W \in \mathbb{R}^+$ be the largest weight of all edges in E: the number of bits in W is $\lceil \log_2 W \rceil$. Therefore, the weight of an edge requires $O(\log W)$ bits and the weight of any path (with no cycle) uses $O(\log(nW))$ bits.

The following Lemma 4 gives the complexity of the protocol *APSP*. Next, the Theorem 5 derives the time and the communication complexity of the algorithm *MDST* from Lemma 4.

Lemma 4 *The All-Pairs Shortest Paths protocol APSP process terminates. It runs in $O(n)$ time and uses $O(nm)$ messages to compute the routing tables at each node of G. Its message size is at most $O(\log n + \log(nW))$.*

PROOF. The protocol *APSP* is almost identical to the well-known distributed Bellman-Ford shortest-paths algorithm (except for the notion of active/inactive nodes). The following definitions are taken from [4].

- Let $S \subseteq V$. A path $[u_0,\dots,u_k]$ is called an S-path if for all i $(0 \le i \le k)$, $u_i \in S$. The S-distance from u to v, denoted $d^S(u,v)$, is the smallest weight of any S-path that joins u to v. When $S = V$, we write $d(u,v) = d^V(u,v)$.

- As a consequence, for all $z \in V$,

(1) If $S' = S \cup \{z\}$, then

$$(\forall u, v \in S) \quad d^{S'}(u,v) \stackrel{def}{=} \min\left(d^S(u,v), d^{S'}(u,z) + d^{S'}(z,v)\right). \tag{2}$$

(2) Let $Neigh_u$ be the set of neighbours of a node $u \in V$,

$$(\forall v \in V)\ d(u,v) \stackrel{def}{=} \begin{cases} 0 & \text{if } u = v \\ \min_{z \in Neigh_u}(\omega(u,z) + d(z,v)) & \text{otherwise.} \end{cases} \tag{3}$$

Since the algorithm is built from the definitions (2) and (3), it does converge to the shortest paths (see [4,18]). Also, since the communication channels are assumed to be FIFO (see [8] and Subsection 1.3), the algorithm process terminates. The above rules ensure that no message in the protocol *APSP* is sent to a terminating node.

Our protocol is based on algorithms which are known to converge in n phases (see [4,18]). For an active node, a phase takes at most two time units in an ABD network (see Subsection 1.3): sending a message to each neighbour and next receiving a message only from all active neighbours). To make the protocol *APSP* process terminate we need an $\langle$**Inactive**$\rangle$ message: in the worst case (for example when G is a line) exchanging $\langle$**Inactive**$\rangle$ messages between nodes takes $O(n)$ time units.

The identity of each node is sent from each active node along each of its adjacent edges. The number of messages sent from every node v is thus $O(n\delta(v))$, where $\delta(v)$ is the degree of v. Inactive nodes simply forward update messages to their inactive neighbours, and they do not increase the message complexity. Therefore, the message complexity of protocol *APSP* is proportional to $2nm = \sum_v n\delta(v)$ [18].

From the rules of the protocol (in Subsection 2.2), adding all $\langle$**Inactive**$\rangle$ messages makes exactly $2m$. Finally, the message complexity of protocol *APSP* is $O(nm)$. Note that each message carries the *ID* of the sending node, the *ID* of the selected node and the distance between both nodes. $\square$

Theorem 5 *The algorithm MDST solves the (MDST) problem for any distributed positively weighted network G in $O(n)$ time. Its communication complexity is $O(nm(\log n + \log(nW)))$ bits, and its space complexity is at most $O(n(\log n + \log(nW)))$ bits (at each node). The number of bits used for the ID of a node is $O(\log n)$, and the weight of a path ending at that node is $O(\log nW)$.*

PROOF. From the previous lemma and Subsection 2.1.5 $\square$

4 Concluding Remarks

Given a positively weighted graph G, our algorithm *MDST* constructs an MDST of G and distributively forwards the control structure over the named network G. This new algorithm is simple and natural. It is also time and message efficient: complexity measures are $O(n)$ and $O(nm)$, respectively, which, in some sense, is "almost" the best achievable (though not optimal) in a distributed setting.

By contrast, the space complexity seems to be far from satisfactory. This is a drawback to the very general assumptions used in the algorithm, especially the assumptions on universal (APSPs) routings in arbitrary network topologies. For example, algorithm *MDST* needs a grand total of $O(n^2(\log n + \log(nW)))$ bits to store all routing tables in the entire network. Now, it was recently shown that reasonable APSP routing schemes require at least $\Omega(n^2)$ bits [11]. This is only a logarithmic factor away from the space complexity of algorithm *MDST*.

References

[1] B. Awerbuch, Complexity of network synchronization, *J. ACM*, 32:804–823, 1985.

[2] B. Awerbuch, Optimal distributed algorithms for minimum weight spanning tree counting, leader election and related problems, *Proc. ACM STOC*, 230–240, 1987.

[3] B. Awerbuch, I. Cidon and S. Kutten, Communication-optimal maintenance of replicated information, *Proc. IEEE FOCS*, 492–502, 1990.

[4] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, 2nd edition, 1992.

[5] M. Bui and F. Butelle, Minimum diameter spanning tree, *Proc. Int. Workshop on Principles of Parallel Computing (OPOPAC'93)*, 37–46, Hermès & Inria, 1993.

[6] F. Butelle, C. Lavault and M. Bui, A uniform self-stabilizing minimum diameter spanning tree algorithm, *Proc. 9th Int. Workshop on Distributed Algorithms (WDAG'95)*, LNCS 972.257-272, Springer-Verlag, 1995

[7] P. M. Camerini, G. Galbiati and F. Maffioli, Complexity of spanning tree problems: Part I, *European J. of Operational Research*, 5:346-352, 1980.

[8] S. Chandrasekaran and S. Venkatesan, A message optimal algorithm for distributed termination detection, *J. of Parallel and Distributed Computing*, 8(3):245-252, 1990.

[9] N. Christophides, *Graph Theory: An algorithmic approach*, Computer Science and Applied Mathematics, Academic Press, 1975.

[10] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook and M. Yung, Maintenance of a minimum spanning forest in a dynamic plane graph, *J. of Algorithms*, 13:33-54, 1992.

[11] P. Fraigniaud and C. Gavoille, Memory requirement for universal routing schemes, *Technical report, LIP 95-05*, ENSL, 1995.

[12] R. G. Gallager, P. A. Humblet and P. M. Spira, A distributed algorithm for minimum weight spanning trees, *ACM TOPLAS*, 5(1):66-77, 1983.

[13] S. L. Hakimi and J. G. Pierce and E. F. Schmeichel, On *p*-centers in networks, *Transportation Sci.*, 12:1-15, 1978.

[14] J.-M. Ho, D. T. Lee, C.-H. Chang and C. K. Wong, Minimum diameter spanning trees and related problems, *SIAM J. on Computing*, 20(5):987-997, 1991.

[15] E. Ihler, G. Reich and P. Widmayer, On shortest networks for classes of points in the plane, *Proc. Int. Workshop on Computational Geometry – Methods, Algorithms and Applications*, Lecture Notes in Computer Science. 103-111, 1991

[16] G. F. Italiano and R. Ramaswani, Maintaining spanning trees of small diameter, *Proc. ICALP'94*, 227-238, 1994.

[17] L. Lamport, Time, clocks and the ordering of events in a distributed system, *CACM*, 21(7):558-565, 1978.

[18] L. Lamport, An assertional correctness proof of a distributed algorithm, *Sci. Computer Programming*, 2:175-206, 1982.

[19] C. Lavault, *Évaluation des algorithmes distribués – analyse, complexité, méthode*, Hermès, 1995.

[20] N. Lynch, *Distributed Algorithms*, Morgan Kauffman, 1996.