# A Uniform Self-Stabilizing Minimum Diameter Spanning Tree Algorithm

Franck Butelle, Christian Lavault, Marc Bui

# A Uniform Self-Stabilizing Minimum Diameter Spanning Tree Algorithm

(Extended Abstract)

Franck Butelle[*], Christian Lavault[†], Marc Bui[‡]

**Abstract**

We present a uniform self-stabilizing algorithm, which solves the problem of distributively finding a minimum diameter spanning tree of an arbitrary positively real-weighted graph. Our algorithm consists in two stages of stabilizing protocols. The first stage is a uniform randomized stabilizing *unique naming* protocol, and the second stage is a stabilizing *MDST* protocol, designed as a *fair composition* of Merlin–Segall's stabilizing protocol and a distributed deterministic stabilizing protocol solving the (MDST) problem. The resulting randomized distributed algorithm presented herein is a composition of the two stages; it stabilizes in $O(n\Delta + \mathcal{D}^2 + n \log\log n)$ expected time, and uses $O(n^2 \log n + n \log W)$ memory bits (where $n$ is the order of the graph, $\Delta$ is the maximum degree of the network, $\mathcal{D}$ is the diameter in terms of hops, and $W$ is the largest edge weight). To our knowledge, our protocol is the very first distributed algorithm for the (MDST) problem. Moreover, it is fault-tolerant and works for any anonymous arbitrary network.

## 1 Introduction

Many computer communication networks require nodes to broadcast information to other nodes for network control purposes, which is done efficiently by sending messages over a spanning tree of the network. Now optimizing the worst-case message propagation delays over a spanning tree is naturally achieved by reducing the diameter to a minimum (see Sect. 1.2); especially in high-speed networks (where the message delay is essentially equal to the propagation delay). However, when communication links fail or come up, and when processors crash or recover, the spanning tree may have to be rebuilt. When the network's topology changes, one option is to perform anew the entire computation of a spanning tree with a minimum diameter from scratch. We thus examine the question of designing an efficient fault-tolerant algorithm, which constructs and dynamically maintains a minimum diameter spanning tree of any anonymous network. The type of fault-tolerance we require is so-called *"self-stabilization"*, which means, informally, that an algorithm must be able to "recover" from any arbitrary transient fault. In this setting, we exhibit a self-stabilizing minimum diameter spanning tree. Our algorithm is asynchronous, it works for arbitrary anonymous network topologies (unique processes ID's are not required), it is uniform (i.e., every process executes the same code; processes are identical), symmetry is broken by randomization, and it stabilizes in efficient time complexity.

[*]Université Paris 10 - 92000 Nanterre. France.
[†]LIPN, CNRS URA 1507, Université Paris-Nord 93430 Villetaneuse. France.
[‡]Université Paris 10 - 92000 Nanterre. France.

## 1.1 Self-Stabilizing Protocols

We consider distributed networks where processes and links from time to time can crash and recover (i.e., dynamic networks), where additionally, when processes recover, their memory may be recovered within an arbitrary inconsistent state (to model arbitrary memory corruption). Despite these faults, we wish the network to be able to maintain and/or to be able to rebuilt certain information about itself (e.g., in this particular case, maintaining a minimum diameter spanning tree). When the intermediate period between one recovery and the next failure is long enough, the system stabilizes.

The theoretical formulation of this model was put forth in the seminal paper of Dijkstra [11], who, roughly, defined the network to be "self-stabilizing" if starting from an *arbitrary* initial state (i.e., after any sequence of faults), the network after some bounded period of time (denoted as *stabilization time*) exhibits a behaviour as if it was started from a good initial state (i.e, stabilizes to a "good" behaviour, or "legitimate state"). Notice that such a formulation does not allow any faults during computation, but allows an arbitrary initial state. Thus, if new faults occur during computation, it is modelled in a self-stabilizing formulation as if it were a *new initial state* from which the network again must recover. In summary, self-stabilization is a very strong fault-tolerance property which covers many types of faults and provides a uniform approach to the design of a variety of fault-tolerant algorithms.

## 1.2 The Minimum Diameter Spanning Tree (MDST) Problem

The use of a control structure spanning the entire network is a fundamental issue in distributed systems and interconnection networks. Since *all* distributed total algorithms have a time complexity $\Omega(D)$, where $D$ is the network diameter, a spanning tree of minimum diameter makes it possible to design a wide variety of time efficient distributed algorithms.

Let $G = (V(G), E(G))$ be a connected, undirected, positively real-weighted graph. The **(MDST) problem** is to find a spanning tree of $G$ of minimum diameter.

In the remainder of the paper, we denote the problem (MDST), *MDST* denotes the protocol and MDST abbreviates the "Minimum Diameter Spanning Tree".

## 1.3 Related Works and Results

The few literature related to the (MDST) problem mostly deals either with graph problems in the Euclidian plane (Geometric Minimum Diameter Spanning Tree), or with the Steiner spanning tree construction (see [19, 20]). The (MDST) problem is clearly a generalization of the (GMDST) problem. Note that when edge weights are real numbers (possibly negative), The (MDST) problem is NP-complete.

Surprisingly, although the importance of having a MDST is well-known, only few papers have addressed the question of how to design algorithms which construct such spanning trees. While the problem of finding and dynamically maintaining a minimum spanning tree has been extensively studied in the literature (e.g., [3, 18] and [4, 17]), there exist no algorithms that construct and maintain dynamically information about the diameter, despite the great importance of this issue in the applications. (Very recently, the distributed (MDST) problem was addressed in [7, 22]). In this paper, we present an algorithm which is robust to transient failures, and dynamically maintains a minimum diameter spanning tree of any anonymous network: a much more efficient (computationally cheaper) solution indeed than recomputing from scratch over and over again.

As opposed to the (quasi-) absence of investigations dealing with the (MDST) problem, and although self-stabilization is quite a new strand of research in distributed computing, a large

number of self-stabilizing algorithms and theoretical related results were proposed during the past few years (e.g., [1, 2, 5, 6, 13, 14, 15, 16, 21, 25, 26]). Due to their features, self-stabilizing protocols were first used in the design of many existing systems (e.g., DECNET protocols [24]).

Our distributed self-stabilizing algorithm is composed of a first uniform stabilizing randomized stage protocol *UN* of *"unique naming"* for arbitrary anonymous networks and of a second stabilizing stage protocol *MDST*, which constructs a MDST. The second stage performs a MDST protocol for *named* networks which results after the first stage stabilizes. This second stage is itself constructed as the *fair composition* [15, 16, 25] of Merlin–Segall's stabilizing distributed routing protocol and a new deterministic protocol for the (MDST) problem. The resulting algorithm $\mathcal{A}$ is thus a composition of the two stages (see Sect. 4.2) to obtain a randomized, uniform, self-stabilizing MDST algorithm $\mathcal{A}$ for general anonymous graph systems.

The complexity of protocols is analyzed by the following complexity measures. The **Time Complexity** of a self-stabilizing algorithm is mainly defined as the time required for stabilization (or *"round complexity"*). More formally, the *stabilization time* of a self-stabilizing deterministic (resp. randomized) algorithm is the maximal (resp. maximal expected) number of rounds that takes the system to reach a legitimate configuration, where the maximum is taken over all possible executions (see the model $\mathcal{M}$ in Sect. 2). The **Space Complexity** of a self-stabilizing algorithm can be expressed as the number of bits required to store the state of each process; i.e., in the message passing model, the maximal size of local memory used by a process. The **Communication Complexity** is measured in terms of the number of bits of the registers; i.e., in the message passing model, the maximal number of bits exchanged by the processes until an execution of the algorithm stabilizes. The time, space and communication complexities of a composed algorithm are the sum of the complexities of the combined protocols.

### 1.3.1 Main contributions of the present paper

- A first stage consisting of a uniform stabilizing randomized *UN* protocol for any arbitrary network $G$, which is an adapted variant of the UN protocol designed in [2]. In model $\mathcal{M}$, our randomized *UN* protocol stabilizes in $O(n \log\log n)$ expected time, with a space complexity $O(n^2 \log n)$.

- An original second stage stabilizing protocol *MDST*, which is designed as the fair composition of Merlin–Segall's stabilizing routing protocol and a new deterministic protocol for the (MDST) problem. The second stage thus constructs a MDST of the named network $G$. In the model $\mathcal{M}$, the protocol *MDST* stabilizes in $O(n\Delta + \mathcal{D}^2)$ time, and its space complexity is $O(n \log n + n \log W)$ bits (where $\Delta$ is the maximum degree of $G$, $\mathcal{D}$ is the diameter in terms of hops and $W$ is the largest edge weight).

- In model $\mathcal{M}$, the resulting randomized composed algorithm $\mathcal{A}$ stabilizes in $O(n\Delta + \mathcal{D}^2 + n \log\log n)$ expected time and uses $O(n^2 \log n + n \log W)$ memory bits. To our knowledge, it appears to be the very first algorithm to *distributively* solve the (MDST) problem. Moreover, our randomized distributed algorithm $\mathcal{A}$ is fault-tolerant and works for any anonymous arbitrary network.

The remainder of the paper is organized as follows: in Sect. 2, we define the formal model $\mathcal{M}$ and requirements for uniform, self-stabilizing protocols, and in Sect. 3 we present the stages of the composed uniform self-stabilizing MDST algorithm $\mathcal{A}$. Section 4.2 and Sect. 5 are devoted to the correctness proof, and to the complexity analysis of stabilizing protocols (*UN*, *MDST*, and algorithm $\mathcal{A}$), respectively. The paper ends with concluding remarks in Sect. 6.

## 2 Model $\mathcal{M}$ (Message Passing)

Formal definitions regarding Input/Output Automata are omitted from this abstract [6, 26].

**IO Automata, Stabilization, Time Complexity** – An Input/Output Automaton (IOA) is a state machine with state transitions which are given labels called *actions*. There are three kinds of actions. The environment affects the automaton through *input actions* which must be responded to in any state. The automaton affects the environment through *output actions*; these actions are controlled by the automaton to only occur in certain states. *Internal actions* only change the state of the automaton without affecting the environment.

Formally, an IOA is defined by a *state* set $S$, an *action* set $L$, a *signature* $Z$ (which classifies $L$ into input, output, and internal actions), a *transition relation* $T \subseteq S \times L \times S$, and a non-empty set of *initial states* $I \subseteq S$. We mostly deal with *uninitialized IOA*, for which $I = S$ ($S$ finite). An action $a$ is said to be *enabled* in state $s$ if there exist $s' \in S$ such that $(s, a, s') \in T$; input actions are always enabled. When an IOA "runs", it produces an execution. An *execution fragment* is an alternating sequence of states and actions $(s_0, a_1, s_1 \ldots)$, such that $(s_i, a_i, s_{i+1}) \in T$ for all $i \geq 0$. An execution fragment is *fair* if any internal or output action which is continuously enabled eventually occurs. An *execution* is an execution fragment which starts with an initial state and is fair. A *schedule* is a subsequence of an execution consisting only of the actions. A *behaviour* is a subsequence of a schedule consisting only of its input and output actions. Each IOA generates a set of behaviours. Finally, let $A$ and $B$ denote two IOA, we say that $A$ *stabilizes* to $B$ if every behaviour of $A$ has a suffix which is also a behaviour of $B$.

For time complexity, we assume that every internal or output action which is continuously enabled occurs in one unit of time. We say that $A$ stabilizes to $B$ in time $t$ if $A$ stabilizes to $B$ and every behaviour of $A$ has a suffix which occurs within time $t$. The *stabilization time* from $A$ to $B$ is the smallest $t$ such that $A$ stabilizes to $B$ in time $t$.

**Network Model** – The model $\mathcal{M}$ is for message passing protocols. The system is a standard point-to-point asynchronous distributed network consisting of $n$ communicating processes connected by $m$ bidirectional links. As usual, the network topology is described by a connected undirected graph $G = (V, E)$, devoid of multiple edges and loop-free. $G$ is defined on a set $V$ of vertices representing the processes and $E$ is a set of edges representing the bidirectional communication links operating between neighbouring vertices: in the sequel, $|V| = n$, and $|E| = m$. We view communication interconnection networks as undirected graphs. Henceforth, we use the terms *graph* (resp. *nodes*/*edges*) and *network* (resp. *processes*/*links*) interchangeably.

Each node and link is modelled by an IOA [6, 26]. A protocol is *uniform* if all processes perform the same protocol and are indistinguishable; i.e., in our model, we do not assume that processes have unique identities (ID's). We drop the adjective "uniform" from now on. The model $\mathcal{M}$ assumes that the messages are transferred on links in FIFO order, and in a finite but unbounded delay. It is also assumed that any non-empty set of processes may start the algorithm (such starting processes are "initiators"), while each non-initiator remains quiescent until reached by some message. In model $\mathcal{M}$, processes have no global knowledge about the system (no structural information is assumed), but only know their neighbours in the network (through the mere knowledge of their ports). In particular, the model $\mathcal{M}$ assumes that nothing is known about the network size $n$ or the diameter $D(G)$ (no upper bound on $n$ or on $D(G)$ is either known). Regarding the use of memory, $\mathcal{M}$ is such that the amount of memory used by the protocols remains bounded, i.e., only a bounded number of messages are stored on each link at any instant. The justification for this assumption is twofold: first, not much can be done with unbounded links in a stabilizing setting [6, 13, 26], and secondly, real channels are inherently bounded anyway. In other words, we model bounded links as unit capacity data links which can store at any given instant at most one circulating message. A link $uv$ from node $u$ to node $v$ is

modelled as a queue $Q_{uv}$, which can store at most one message from some message alphabet $\Sigma$ at any instant time. The external interface to the link $uv$ includes an input action $\textsc{Send}_{uv}(m)$ ("send message $m$ from $u$"), an output action $\textsc{Receive}_{uv}(m)$ ("deliver message $m$ at $v$"), and an output action $\textsc{Free}_{uv}$ ("the link $uv$ is currently free"). If a $\textsc{Send}_{uv}(m)$ occurs when $Q_{uv} = \emptyset$, the effect is that $Q_{uv} = \{m\}$; when $Q_{uv} = \emptyset$, $\textsc{Free}_{uv}$ is enabled. If a $\textsc{Send}_{uv}(m)$ occurs when $Q_{uv} \neq \emptyset$, there is no change of state. Note that by the above timing assumptions, a message stored in a link will be delivered in one unit of time.

We refer to [6] for detailed and formal definitions of the notions of *queued node automaton*, *network automaton for a graph G*, and similarly for the notions of *internal reset* and *stabilization by local checking and global reset*. (See Sect. 4.2 for the definition of local checkability and the statement of the two main theorems used in the correctness proof of the algorithm).

# 3 The Algorithm

Let $G = (V(G), E(G))$ be a connected, undirected, positively real-weighted graph, where the weight of an edge $e = uv \in E(G)$ is given by $\omega_{uv}$. In the remainder of the paper, we use the graph theoretical terminology and notation. The weight of a path $[u_0, \ldots, u_k]$ of $G$ ($u_i \in V(G)$) is defined as $\sum_{i=0}^{k-1} \omega_{u_i u_{i+1}}$. For all nodes $u$ and $v$, the *distance* from $u$ to $v$, denoted $d_G(u, v)$, is the lowest weight of any path length from $u$ to $v$ in $G$ ($\infty$ if no such path exists). The distance $d_G(u, v)$ represents the *shortest path* from $u$ to $v$, and the largest (maximal) distance from node $v$ to all other nodes in $V(G)$, denoted $s_G(v)$, is the *separation* of node $v$: viz. $s_G(v) = \max_{u \in V(G)} d_G(u, v)$ [10]. $D(G)$ denotes the diameter of $G$, defined as $D(G) = \max_{v \in V(G)} s_G(v)$, and $\mathcal{D}(G)$ the diameter in terms of hops. $R(G)$ denotes the radius of $G$, defined as $R(G) = \min_{v \in V} s_G(v)$. $\Psi_G(u)$ represents a shortest-paths tree (SPT) rooted at node $u$: $(\forall v \in V(G))\ \ d_{\Psi_G(u)}(u, v) = d_G(u, v)$. The set of all SPT's of $G$ is then denoted $\Psi(G)$. The name of the graph will be omitted when it is clear from the context.

## 3.1 A High-Level Description

### 3.1.1 Unique Naming Protocol

The *unique naming* protocol solves the (UN) problem, where each process $u$ must select one ID distinct from all other processes'. The protocol executes propagation of information (propagation of the ID of process $u$) and feedback ($u$ collects the ID's of all other processes): i.e., a "PIF" protocol. Our randomized stabilizing protocol *UN* is a variant of the memory adaptive UN PIF protocol presented in [2] and slightly differs in the following respects. First, our results hold for the message passing model $\mathcal{M}$, even though they can easily be transposed in the link register model (and *vice versa*: the results in [2] can easily be extended to the message passing model). Next, we do not use the ranking phase designed in the original protocol, but a simple ID's conflict checking phase. Besides, our maximum estimate for the size of the network is arbitrarily chosen to be $\leq \lg n$ (see the proof of Theorem 5.1 in [8]), instead of $n^{1/2} - n^{1/3}$ in [2]. Note that the model $\mathcal{M}$ assumes that nothing is known about $n$ or $D(G)$ (not even an upper bound), therefore, the UN Monte-Carlo protocol in [2] *cannot* be turned into a randomized Las Vegas protocol (e.g., a protocol solving the (UN) problem with probability 1).

Due to the lack of space, we do not give a detailed description of our protocol *UN* herein. A full description of the three phases executed in the protocol can be found in [2] (for the original version) and in [8] (for our own variant). However, for better understanding of self-stabilization (showed in Sect. 4.2), let us just point out the behaviour of protocol *UN* in phase 3. Each process in phase 3 repeatedly broadcasts a message with its ID. At the end of each broadcast, if $u$ detects

a conflict, it initiates a Reset. In addition, $u$ collects the ID's of all other processes (provided by feedback) and checks that all processes have unique ID's. The variable $IDList$ contains the list of ID's of the visited processes. At the beginning of each broadcast, it is set to the initiator's ID; each visited process attaches its own ID to the list before forwarding it to its neighbours. After stabilization, every process remains forever in phase 3.

### 3.1.2 Construction of a MDST

The definition of separation must be generalized to *"dummy nodes"* (so-called in contrast to actual vertices of $V$). Such a fictitious node may possibly be inserted on any edge $e \in E$. Thus, let $e = uv$ be an edge of weight $\omega_{uv}$, a dummy node $\gamma$ inserted on $e$ is defined by specifying the weight $\alpha$ of the segment $u\gamma$. According to the definition, the separation $s(\gamma)$ of a *general node* $\gamma$, whether it is an actual vertex in $V$ or a dummy node, is clearly given by: $s(\gamma) = \max_{z \in V} d(\gamma, z)$. A node $\gamma^*$ such that $s(\gamma^*) = \min_\gamma s(\gamma)$ is called an *absolute center* of the graph. Recall that $\gamma^*$ always exists in a connected graph, and that is not unique in general.

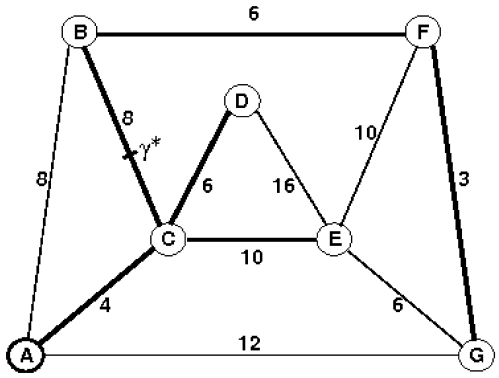


Figure 1: Example of a MDST $T^*$ ($D(G) = 22$ and $D(T^*) = 27$)

Similarly, the definition of $\Psi(u)$ is also generalized so as to take these dummy nodes into account. Finding a MDST actually amounts to search for an absolute center $\gamma^*$ of $G$, and the SPT rooted at $\gamma^*$ is then a MDST of $G$. Such is the purpose of the following Lemma:

**Lemma 3.1** [9] *The (MDST) problem for a given graph $G$ is (polynomially) reducible to the problem of finding an absolute center of $G$.*

### 3.1.3 Computation of an absolute center of a graph

According to the results in [10], we use the following Lemma to find an absolute center of $G$.

**Lemma 3.2** *Let $G = (V, E)$ be a weighted graph. An absolute center $\gamma^*$ of $G$ is constructed as follows:*

*(i) On each edge $e \in E$, find a general node $\gamma_e$ of minimum separation.*

*(ii) Among all the above $\gamma_e$'s, $\gamma^*$ is a node achieving the smallest separation.*

**Proof:** (the proof is constructive)

*(i)* This first step is performed as follows: for each edge $e = uv$, let $\alpha = d(u, \gamma)$. Since the distance $d(\gamma, z)$ is the length of either a path $[\gamma, u, \ldots, z]$, or a path $[\gamma, v, \ldots, z]$,

$$s(\gamma) = \max_{z \in V} d(\gamma, z) = \max_{z \in V} \min\{\alpha + d(u, z), \omega_{uv} - \alpha + d(v, z)\}. \quad (1)$$

If we plot $f_z^+(\alpha) = \alpha + d(u, z)$ and $f_z^-(\alpha) = -\alpha + \omega_{uv} + d(v, z)$ in Cartesian coordinates for fixed $z = z_0$, the real-valued functions $f_{z_0}^+(\alpha)$ and $f_{z_0}^-(\alpha)$ (separately depending on $\alpha$ 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)) is thus found by taking the *convex cone* of $(S_1)_{z_0}$ and $(S_{-1})_{z_0}$. By repeating the above process for each node $z \in V$, all convex cones of segments $(S_1)_{z \in V}$ and $(S_{-1})_{z \in V}$ are clearly obtained (see Fig. 2).

Now we can draw the *upper boundary* $B_e(\alpha)$ ($\alpha \in [0, \omega_e]$) of all the above convex cones of segments $(S_1)_{z \in V}$ and $(S_{-1})_{z \in V}$. $B_e(\alpha)$ is thus a curve made up of piecewise linear segments, which passes through several local minima (see Fig. 2). The point $\gamma$ achieving the smallest minimum value (i.e., the global minimum) of $B_e(\alpha)$ represents the absolute center $\gamma_e^*$ of the edge $e$.

*(ii)* By definition of the $\gamma_e^*$'s, $\min_\gamma s(\gamma) = \min_{\gamma_e^*} s(\gamma_e^*)$, and $\gamma^*$ achieves the smallest separation. Therefore, an absolute center of the graph is found at any point where the minimum of all $s(\gamma_e^*)$'s is attained. □

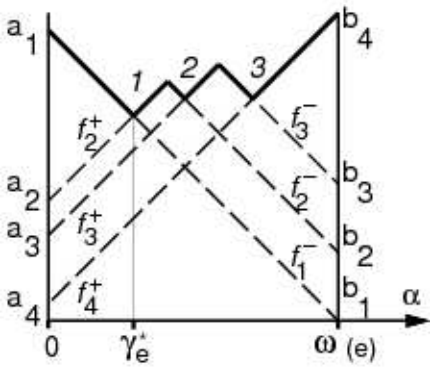


Figure 2: Example of an upper boundary $B_e(\alpha)$

By Lemma 3.2, we may consider this method from an algorithmic viewpoint. For each $e = uv$, let $\mathcal{C}_e$ be the set of pairs $\{(d_1, d_2)\ /\ (\forall z \in V)\ d_1 = d(u, z), d_2 = d(v, z)\}$ Now, a pair $(d_1', d_2')$ is said to *dominate* a pair $(d_1, d_2)$ iff $d_1 \leq d_1'$, and $d_2 \leq d_2'$ (viz. the convex cone of $(d_1', d_2')$ is over the convex cone of $(d_1, d_2)$). Any such pair $(d_1, d_2)$ will be ignored when it is dominated by another pair $(d_1', d_2')$.

Notice that the local minima of the upper boundary $B_e(\alpha)$ (numbered from 1 to 3 in Fig. 2) are located at the intersection of segments $f_i^-(\alpha)$ and $f_{i+1}^+(\alpha)$, when all dominated pairs are removed. If we sort the set $\mathcal{C}_e$ in descending order with respect to the first term of each remaining pair $(d_1, d_2)$, we thus obtain the list $L_e = ((a_1, b_1), \ldots, (a_{|L_e|}, b_{|L_e|})$ consisting in all such remaining ordered pairs. Hence, the smallest minimum of $B_e(\alpha)$ for a given edge $e$ clearly provides an absolute center $\gamma_e^*$. (See Procedure `Gamma_star(e)` in Sect. 3.2). By Lemma 3.2, once all the $\gamma_e^*$'s are computed, an absolute center $\gamma^*$ of the graph is obtained. By Lemma 3.1, finding a MDST of the graph reduces to the problem of computing $\gamma^*$.

### 3.1.4 All-Pairs Shortest-Paths Protocol (APSP)

In the previous paragraph, we consider distances $d(u, z)$ and $d(v, z)$, for all $z \in V$ and each edge $e = uv$. Such distances must be computed by a failsafe distributed routing protocol, e.g., Merlin–Segall's APSP protocol designed in [23].

The justification for this choice is threefold. First, shortest paths to each destination $v$ are computed by executing the protocol independently for each $v$. Thus, an essential property of Merlin–Segall's algorithm is that the routing tables are cycle-free at any time (Property (a) in [23]). Next, the protocol is also adapted to any change in the topology and the weight of edges (Property (b)). Finally, the protocol converges in dynamic networks and is indeed self-stabilizing (Property (c)). (See Lemma 4.1).

## 3.2 A Formal Description

Assume the list $L_e$ defined above (in Paragraph 3.1.3) to be already constructed (for example with a heap, whenever the routing tables are computed), the following procedure computes the value of $\gamma_e^*$ for any fixed edge $e$.

**Procedure** `Gamma_star(`$e$`)`

**var** $min, \alpha$ : real  **Init** $min \leftarrow +\infty$ ; $\alpha \leftarrow 0$ ;
**For** i=1 to $|L_e|$ **do**
  *compute the intersection* $(x, y)$ *of segments* $f_i^-$ *and* $f_{i+1}^+$ *:*
  $x = \frac{1}{2}(\omega_e - a_i + b_{i+1})$ ; $y = \frac{1}{2}(\omega_e + b_{i+1} + a_i)$
  **if** $y < min$ **then** $min \leftarrow y$ ; $\alpha \leftarrow x$ ;
**Return**($\alpha$,$min$)

The distributed protocol *MDST* finds a MDST of an input graph $G = (V, E)$ by computing the diameter of the SPT's for all nodes. Initially, an edge weight $\omega_{uv}$ is only known by its two endpoints $u$ and $v$. In the first stage, the randomized, stabilizing protocol *UN* provides each process $u$ with its unique ID, denoted $ID_u$ (see Sect. 3.1.1).

**Protocol** $MDST$ **(for process** $u$**)**

**Type** elt : **record** $alpha\_best$, $upbound$ : real ; $ID_1$, $ID_2$: integer **end** ;
**Var** $\Lambda$ : set of elt ; $\varphi$, $\varphi_u^*$ : elt ; $D$, $R$, $\alpha$, $localmin$ : real ;
$d_u$ : array of weights ; *(* $d_u[v]$ estimates $d(u,v)$ *)*

1. **For all** $v \in V$
   Compute $d_u[v]$, $D$ and $R$ ; *(* by Merlin–Segall's protocol *)*
2. $\varphi.upbound \leftarrow R$ ;
3. **While** $\varphi.upbound > D/2$ **do for** any edge $uv$ s.t. $ID_v > ID_u$
   (a) $(\alpha, localmin) \leftarrow$ `Gamma_star(`$uv$`)` ;
   (b) **If** $localmin < \varphi.upbound$ **then** $\varphi \leftarrow (\alpha, localmin, ID_u, ID_v)$ ;
4. $\Lambda \leftarrow \{\varphi\}$ ;
5. **Receive** $\langle\varphi\rangle$ from all sons of $u$ in $\Psi(r)$
   ($r$ is s.t. $ID_r = \min_{v \in V}\{ID_v\}$) ; $\Lambda \leftarrow \Lambda \cup \{\varphi\}$ ;
6. Minimum finding:
   (a) Compute $\varphi_u^*$ s.t. $\varphi_u^*.upbound = \min\limits_{\varphi \in \Lambda} \varphi.upbbound$ ;
   **Send** $\langle\varphi_u^*\rangle$ to father in $\Psi(r)$ ;
   (b) **If** $ID_u = ID_r$ **then** upon reception of $\langle\varphi\rangle$ from all sons of $r$, $r$ forwards $\langle\varphi_u^*\rangle$ to all other nodes.

**Remark** In order to complete self-stabilization, the deterministic protocol $MDST$ must be repeatedly executed .

A *sequential* algorithm for the (MDST) problem may also be derived from the above protocol, since $\Psi(\gamma)$ is then a MDST of $G$, where $\gamma$ is the general node s.t. $s(\gamma) = upbound$.

**Improvements:** In practice, some improvements in protocol $MDST$ can easily be carried out. Indeed, reducing the enumeration of dummy nodes may be done by discarding several edges of $G$ from the exploration. To be able to discard an edge, we only need to know bounds on the minimum diameter $D^*$ of all spanning trees of $G$. Note that the lower bound on $D^*$ is obviously $D(G)$, and that $D^*$ is also bounded from above by the minimum diameter taken over all SPT's, viz. $D^* \leq \min_{T \in \Psi(G)} D(T)$. In the example of Fig. 1, such improvements lead to discard from the exploration the edges EF, AB, AC, BF, CD, DE, EG, FG. (See [8]).

# 4 Correctness

## 4.1 Self-Stabilization

Fix a network automaton $\mathcal{N}$ for a given graph $G$, the definition of local checkability is stated as follows [6].

**Definition 4.1** *Let $\mathcal{L} = \{LP_{uv}\}$ be a set of local predicates, and let $\psi$ be any predicate of $\mathcal{N}$. A network automaton $\mathcal{N}$ is locally checkable for $\psi$ using $\mathcal{L}$ if the following conditions hold.*

*(i) For all states $s \in S(\mathcal{N})$, if $s$ satisfies $LP_{uv}$ for all $LP_{uv} \in \mathcal{L}$, then $s \in \psi$.*

*(ii) There exists $s \in S(\mathcal{N})$ such that $s$ satisfies $LP_{uv}$ for all $LP_{uv} \in \mathcal{L}$.*
*(iii) Each $LP_{uv} \in \mathcal{L}$ is stable: for all transitions $(s, a, s')$ of $\mathcal{N}$, if $s$ satisfies $LP_{uv}$ then so does $s'$.*

The main theorem in [6] is about self-stabilization by local checking and global reset. Roughly, it shows that any protocol which is locally checkable for some global property can be transformed into an equivalent protocol, which stabilizes to a variant of the protocol in which the global property holds in its initial state. This transformation increases the time complexity by an overhead given in [6, Theorem 10].

Also recall the fundamental Theorem 4.1 which states the fair composition of two stabilizing protocols $P_1$ and $P_2$ [15].

**Theorem 4.1** *If the four conditions hold,*
*(i) protocol $P_1$ stabilizes to $\psi_1$;*
*(ii) protocol $P_2$ stabilizes to $\psi_2$ if $\psi_1$ holds;*
*(iii) protocol $P_1$ does not change variables used by $P_2$ once $\psi_1$ holds; and,*
*(iv) all executions are fair w.r.t. both $P_1$ and $P_2$,*
*then the fair composition of $P_1$ and $P_2$ stabilizes to $\psi_2$.*

## 4.2 Correctness Proof

Let $\psi$ be a predicate over the variables of protocol *UN*, and $\psi'$ a predicate over the variables of protocol *MDST* (see Sect. 2). Now, protocol *MDST* is the fair combination of two subprotocols. The first protocol uses Merlin–Segall's APSP routing algorithm (see Sect. 3.1.4), while the second subprotocol deterministically computes the value $\gamma^*$ (see Sect. 3.1.3). Hence, the local predicates $LP_{uv}$ and $LP'_{uv}$ corresponding to the predicates $\psi$ and $\psi'$, respectively, are defined by

$$\begin{aligned} LP_{uv} \quad \equiv \quad & \{(\forall\, ID_i, ID_j \in IDList_u)\ \ i \neq j \implies ID_i \neq ID_j\} \\ & \wedge\ \ \{(\forall\, ID_i, ID_j \in IDList_v)\ \ i \neq j \implies ID_i \neq ID_j\} \\ & \wedge\ \ (u \textit{ and } v \textit{ are both in phase } 3) \text{ for predicate } \quad \psi \equiv (\forall uv \in E)\ \ LP_{uv}, \end{aligned}$$

where the variable $IDList$ is defined in Sect. 3.1.1. And, similarly,

$$LP'_{uv} \equiv (d_u[v] < +\infty)\ \wedge\ (d_v[u] < +\infty), \text{ for predicate } \quad \psi' \equiv (\forall uv \in E)\ \ LP'_{uv}.$$

Note that this does not mean that the estimate values $d_u[v]$ are exact, but that they are not too bad. Of course, if some distances $d_u[v]$ are wrong, it may cause the construction of a MDST to fail. However, the routing protocol is self-stabilizing, and after a while the estimate distances shall be correct and a MDST will be found.

**Lemma 4.1** *Let $\mathcal{L} = \{LP_{uv}\}$ be the set of local predicates over the variables of the randomized protocol UN. A network automaton $\mathcal{N}$ is locally checkable for $\psi$ using $\mathcal{L}$.*

**Proof:** (By Definition 4.1). Condition *(i)* clearly holds by the definition of $\psi$. Condition *(ii)* holds for a state $s \in S(\mathcal{N})$ such that processes ID's are all distinct in phase 3.

Now suppose $s \in S(\mathcal{N})$ satisfies $LP_{uv}$. In the case when no failures occur, $u$ and $v$ obviously remain in phase 3 by construction of protocol *UN*. In the case when nodes recoveries occur (with arbitrary ID's), $u$ and $v$ are able to detect conflicts and if necessary they initiate a Reset. After a while, each process (and especially $u$ and $v$) returns to phase 3 with one unique ID. Therefore, condition *(iii)* holds. □

**Lemma 4.2** *Let $\mathcal{L}' = \{LP'_{uv}\}$ be the set of local predicates over the variables of Merlin–Segall's APSP protocol. A network automaton $\mathcal{N}$ is locally checkable for $\psi'$ using $\mathcal{L}'$.*

**Proof:** (By Definition 4.1). Condition *(i)* clearly holds by the definition of $\psi'$. Since $G$ is connected, there exists a path $[u, \ldots, v]$ such that the distance $d_G(u, v)$ is finite. Hence, condition *(ii)* holds for the corresponding state $s \in S(\mathcal{N})$. Finally, condition *(iii)* clearly holds by convergence of Merlin–Segall's routing protocol. (See [23, Property (c)], and Sect. 3.1.4). □

Recall that $\varphi^*_u.upbound$ denotes the best value of $s(\gamma^*)$ computed so far at node $u$. We show now that both protocols *MDST* and $\mathcal{A}$ stabilize to the desired postcondition $\theta$ defined by:

$$\theta \equiv (\forall u \in V)\ \ \varphi^*_u.upbound = s(\gamma^*).$$

The local predicate $LP''_{uv}$ corresponding to $\theta$ is defined by:

$$LP''_{uv} \equiv \varphi^*_u.upbound = s(\gamma^*)\ \wedge\ \varphi^*_v.upbound = s(\gamma^*).$$

**Lemma 4.3** *Assume processes ID's are all distinct, the protocol MDST stabilizes to $\theta$.*

**Proof:** (Sketch) First, protocol *MDST* is locally checkable for $\theta$ using the set $\mathcal{L}'' = \{LP''_{uv}\}$. By Definition 4.1, conditions *(i)* and *(ii)* clearly hold. Condition *(iii)* derives from the fact that Merlin–Segall's protocol stabilizes to $\psi'$, while the computation of $\gamma^*$ is deterministic. Consequently, protocol *MDST* is locally checkable and stabilizes to $\theta$ by [6, Theorem 10]. □

**Theorem 4.2** *The randomized algorithm $\mathcal{A}$ stabilizes to $\theta$ with probability 1.*

**Proof:** The following conditions hold.

*(i)* By Lemma 4.1 and [6, Theorem 10], protocol *UN* stabilizes to $\psi$ with probability 1.

*(ii)* By Lemma 4.3, protocol *MDST* stabilizes to $\theta$ if $\psi$ holds.

*(iii)* By construction, protocol *UN* does not change variables used by *MDST* once $\psi$ holds.

*(iv)* Since protocol *MDST* terminates, there are only finitely many executions of *MDST* between two executions of *UN*. The protocol *UN* stabilizes to $\theta$ with probability 1 and since $\theta$ is true, each ID remains unchanged, and so does the computation of $\gamma^*$. Therefore, all executions are fair w.r.t. to both *UN* and *MDST*.

By Theorem 4.1, algorithm $\mathcal{A}$ which is the fair composition of *UN* and *MDST* stabilizes to $\theta$ with probability 1. □

# 5 Analysis

## 5.1 Protocol *UN*

The three phases executed in protocol *UN* are described in [2, 8]. (See Sect. 3.1.1).

**Lemma 5.1** *Each Reset lasts at most $2D + n$ rounds. If two processes have the same ID, then within at most $O(n)$ rounds some process in the network will order a Reset. After a Reset, it takes the system $4n$ rounds either to perform global memory adaptation, or to complete another Reset.*

Note that the maximum number of rounds needed for the completion of phases 1 and 2 is exactly $4n$.

**Lemma 5.2** *If $n$ processes choose random ID's from the set $[N] = \{1, \ldots, N\}$, where $N \geq n^2/\epsilon$, all ID's will be unique with probability $p > 1 - \epsilon$, for all $0 < \epsilon < 1$.*

**Proof:** The probability $p$ that all processes randomly choose distinct ID's is

$$p \;=\; \frac{N(N-1)\cdots(N-n+1)}{N^n} \;=\; \prod_{i=1}^{n-1}(1 - i/N).$$

Assuming that $n/N \leq 1/2$, or $N \geq 2n$ yields

$$p \;>\; \prod_{i=1}^{n-1} e^{-2i/N} \;>\; e^{-n^2/N}.$$

Since $(\forall\, 0 < \epsilon < 1)\ e^{-\epsilon} > 1 - \epsilon$, we have that $p > 1 - \epsilon$ when $N \geq n^2/\epsilon$. Hence, it is sufficient to randomly select the $n$ identities from the set $[N]$, with $N \geq n^2/\epsilon$, in which case the identities are all distinct with probability $> 1 - \epsilon$, for fixed $0 < \epsilon < 1$. □

**Lemma 5.3** *If after a Reset there exist $n' < n$ distinct ID's in the network, then a Reset is initiated by the end of phase 2 with probability $\geq 1 - 2^{n'-n}$.*

**Theorem 5.1** *Let $\Delta$ be the maximum degree of the network. Starting from any state, the probability that the system will stabilize in $O\left(n(1 + \log\log n - \log\log(\Delta + 1))\right)$ rounds is $\geq 1 - \delta$, for some constant $0 < \delta < 1$ which does not depend on the network. The expected number of rounds until protocol UN stabilizes is $O(n \log\log n)$. The maximal memory size used by each process in any execution of protocol UN is at most $O(n^2 \log n)$ bits.*

## 5.2 Protocol *MDST*

**Lemma 5.4** *The time complexity of protocol MDST is at most $O(n\Delta + \mathcal{D}^2)$, and its space complexity is $O(n \log n + n \log W)$ bits, where $W$ is the largest edge weight.*

**Proof:** It is shown in [23] that after $i$ update rounds, all shortest paths of at most $i$ hops have been correctly computed, so that after at most $\mathcal{D}$ rounds, all shortest paths to node $u$ are computed. Shortest paths to each destination are computed by executing the protocol independently for each destination. Since a round costs $O(\mathcal{D})$ time, the stabilization time of Merlin–Segall's protocol is $O(\mathcal{D}^2)$. Now, the computation of $\gamma^*$ requires a minimum finding over a tree (viz., $O(n)$) and local computations on each adjacent edge of $G$ (viz., $O(\Delta)$, where $\Delta$ is the maximum degree). Hence, the stabilization time of the protocol *MDST* is $O(n\Delta + \mathcal{D}^2)$.

Finally, $O(n \log n + n \log W)$ space complexity is needed to maintain global routing tables. □

Note that since $\mathcal{D} \leq D \leq W\mathcal{D}$, the "hop time complexity" used above is more accurate.

## 5.3 Complexity Measures of Algorithm $\mathcal{A}$

The following theorem summarizes our main result, and its proof follows from the previous Lemma.

**Theorem 5.2** *Starting from any state, the probability that algorithm $\mathcal{A}$ will stabilize is $\geq 1 - \delta$, for some constant $0 < \delta < 1$ which does not depend on the network. Recall $\mathcal{D}$ be the diameter of $G$ in terms of hops, $\Delta$ the maximum degree, and $W$ the largest edge weight. The expected time complexity of $\mathcal{A}$ is $O(n\Delta + \mathcal{D}^2 + n \log\log n)$, and its space complexity is at most $O(n^2 \log n + n \log W)$ bits.*

Since the number of messages required in Merlin–Segall's protocol is at most $O(n^2m)$, the communication complexity of $\mathcal{A}$ is $O(n^2mK)$ bits (where $K = O(\log n + \log W)$ bits is the largest message size).

# 6 Concluding Remarks

We proposed a uniform self-stabilizing algorithm for distributively finding a MDST of a positively weighted graph. Our algorithm is new. It works for arbitrary anonymous networks topologies, symmetry is broken by randomization; it stabilizes in $O(n\Delta + \mathcal{D}^2 + n \log\log n)$ expected time, and requires at most $O(n^2 \log n + n \log W)$ memory bits. The assumptions of our model $\mathcal{M}$ are quite general, and in some sense, the algorithm might be considered reasonably efficient in such a setting (even though the communication complexity appears to be the weak point of such algorithms). Whatsoever, the stabilization complexities can be improved in terms of time and space efficiency by restricting the model's assumptions and using the very recent results proposed in [12] and [5]. First, the randomized uniform self-stabilizing protocol presented in [12] provides each (anonymous) process of a uniform system with a distinct identity. This protocol for unique naming uses a predefined fixed amount of memory and stabilizes within $\Theta(D)$ expected time (where $D$ is the diameter of the network). Secondly, following [5], we may restrict our model and assume that a pre-specified bound $B(D)$ on the diameter $D$ is known. In $O(D)$ time units, the stabilizing protocol in [5] produces a shortest paths tree rooted at the minimal ID node of the network; in addition, the complexity of the space requirement and messages size is $O(\log B(D))$. In this restricted model (i.e., assuming the knowledge of an upper bound on $D$), the fair composition of the two protocols yields a randomized uniform self-stabilizing algorithm which finds a MDST with stabilization time (at most) $O(n)$ and space complexity $O(\log B(D))$. In this setting, the fact that the space complexity does *not* depend on $n$ makes the solution more adequate for dynamic networks.

# References


[1] Y. Afek, G. Brown. Self-stabilization of the alternating-bit protocol, *Proc. Symp. Reliable Distr. Syst.*, pages 80-83, 1989.

[2] E. Anagnostou, R. El-Yaniv, V. Hadzilacos. Memory adaptative self-stabilizing protocols, *Proc. WDAG*, pages 203-220, 1992.

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

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

[5] B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, G. Varghese. Time optimal self-stabilizing synchronization, *Proc. ACM STOC*, 1993.

[6] B. Awerbuch, B. Patt-Shamir, G. Varghese, S. Dolev. Self-stabilization by local checking global reset, *Proc. WDAG*, pages 326-339, 1994.

[7] M. Bui F. Butelle. Minimum diameter spanning tree, *OPOPAC Proc. Int. Workshop on Principles of Parallel Computing*, pages 37-46. Hermès & Inria, 1993.

[8] F. Butelle, C. Lavault, M. Bui. A uniform self-stabilizing minimum diameter spanning tree algorithm, *RR. 95-07, LIPN, University of Paris-Nord*, May 1995.

[9] P. M. Camerini, G. Galbiati, F. Maffioli. Complexity of spanning tree problems: Part I, *Europ. J. Oper. Research*, **5**:346-352, 1980.

[10] N. Christophides. *Graph Theory: An algorithmic approach*, Computer Science Applied Mathematics, Academic press, 1975.

[11] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control, *CACM,* ***17***, (11):643-644, 1974.

[12] S. Dolev. Optimal Time Self-Stabilization in Uniform Dynamic Systems, *Proc. 6th Int. Conf. on Parallel Distributed Computing Systems*, 1994.

[13] S. Dolev, A. Israeli, S. Moran. Resource bounds on self-stabilizing message driven protocols, *Proc. ACM PODC*, 1991.

[14] S. Dolev, A. Israeli, S. Moran. Uniform dynamic self-stabilizing leader election Part 1: Complete graph protocols, *Proc. WDAG*, 1991.

[15] S. Dolev, A. Israeli, S. Moran. Self-stabilization of dynamic systems assuming read/write atomicity, *Distributed Computing*, **7**(1):3-16, 1993.

[16] S. Dolev, A. Israeli, S. Moran. Uniform self-stabilizing leader election Part 2: General graph protocol, *Technical report, Technion, Israel*, March 1995.

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

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

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

[20] E. Ihler, G. Reich, P. Wildmayer. On shortest networks for classes of points in the plane, *Int. Workshop on Comp. Geometry - Meth., Algo. Applic.*, LNCS: 103-111, 1991.

[21] S. Katz, K. J. Perry. Self-stabilizing extensions for message-passing systems, *Distributed Computing*, 7:17-26, 1993.

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

[23] P. M. Merlin A. Segall. A failsafe distributed routing protocol, *IEEE Trans. Comm.*, COM-27(9):1280-1287, Sept. 1979.

[24] R. Perlman. Fault-tolerant broadcast of routing information, *Computer Networks*, **7**:395-405, 1983.

[25] S. K. Shukla, D. Rosenkrantz, S. S. Ravi. Observations on self-stabilizing graph algorithm for anonymous networks, *Technical report, University of Albany, NY*, 1995.

[26] G. Varghese. Self-stabilization by counter flushing, *Proc. ACM PODC*, pages 244-253, 1994.