Journal of Graph Algorithms and Applications
http://www.cs.brown.edu/publications/jgaa/
vol. 5, no. 5, pp. 39–57 (2001)

Finding All the Best Swaps of a Minimum Diameter Spanning Tree Under Transient Edge Failures

Enrico Nardelli *Guido Proietti*
Dipartimento di Matematica Pura ed Applicata
Università di L'Aquila, Italy, and
Istituto di Analisi dei Sistemi e Informatica, CNR, Roma, Italy
http://w3.dm.univaq.it/∼{nardelli,proietti}
{nardelli,proietti}@univaq.it

Peter Widmayer
Institut für Theoretische Informatik
ETH Zentrum, Zürich, Switzerland
http://www.inf.ethz.ch/personal/widmayer
widmayer@inf.ethz.ch

Abstract

In network communication systems, frequently messages are routed along a minimum diameter spanning tree (MDST) of the network, to minimize the maximum delay in delivering a message. When a transient edge failure occurs, it is important to choose a temporary replacement edge which minimizes the diameter of a new spanning tree. Such an optimal replacement is called a *best swap*. As a natural extension, the *all-best-swaps (ABS) problem* is the problem of finding a best swap for every edge of the MDST. Given a weighted graph $G = (V, E)$, where $|V| = n$ and $|E| = m$, we solve the ABS problem in $O(n\sqrt{m})$ time and $O(m)$ space, thus improving previous bounds for $m = o(n^2)$. We also show that the diameter of the tree resulting from a best swap is at most 5/2 as long as the diameter of a MDST recomputed from scratch.

Communicated by Takao Nishizeki, Roberto Tamassia, and Dorothea Wagner: submitted January 1999; revised February 2000 and December 2000.

Work supported by the EU TMR Grant CHOROCHRONOS and by the Swiss National Science Foundation. A preliminary version of this paper was presented to the 6th European Symposium on Algorithms (ESA'98), Venice, Italy, 1998.

1 Introduction

For communication networks, it is important to remain operational even if individual network components fail. In the past few years, the ability of a network to survive a failure (its *survivability*) has been studied intensely (an excellent survey paper on survivable networks is [5]). From the practical side, this has largely been a consequence of the replacement of metal wire meshes by fiber optic networks: Their extremely high bandwidth makes it economically attractive to make networks as sparse as possible. A sparse network, however, is less likely to survive a link (or node) failure than a mesh with a multitude of connections that can be used as detours. Therefore, it is important for sparse networks to also take survivability into account from the very beginning.

In the extreme, a sparse network might coincide with a spanning tree of some underlying weighted graph G of n nodes and m edges. Such a spanning tree generally minimizes some objective function defined according to some network functionality that one wants to optimize. A typical example is that of the *minimum spanning tree* (MST), i.e., a spanning tree whose total weight is minimum among all the spanning trees of G. For tree-like communication networks, however, even a single link failure will affect the service. Notice that these failures should be considered *temporary* (or *transient*), since usually the disrupted link will be restored reasonably quickly. Hence, it makes sense to guarantee the network functionality during this short recovering time by simply reconnecting the two disconnected components through a *single* replacement link. The *emergency network* thus obtained will not be, in general, a best possible spanning tree (according to the chosen objective function) of the graph G now deprived of the failed link. However, it offers the considerable advantage of being readily operative at a low cost, since it will reuse almost all of both the physical network and the communication protocol previous to the failure. The alternative of rebuilding a new optimal spanning tree, which might possibly be totally different from the previous one, would indeed be excessive and disproportionate, considering its short life.

On the theoretical side, this scenario gave rise to an interesting family of problems on graphs. The fundamental question we want to answer to is the following: *Given a weighted, undirected and 2-edge connected graph $G = (V, E)$, and a spanning tree $T = (V, E_T)$ of G which minimizes some objective function $\phi(G, T)$, and given an edge $e \in E_T$, find an edge $e' \in E$ such that the* swap tree *$T_{e/e'}$ obtained by swapping e with e' is a spanning tree of $G - e = (V, E \setminus \{e\})$, and $\phi(G - e, T_{e/e'})$ is minimum over all the possible swap trees.* Such an edge e' will be referred to in the following as a *best swap edge* for e with respect to ϕ. To make things more concrete, let us go back to the sample case where T is a MST of G. Here, it is easy to see that a best swap edge is simply an edge of minimum weight connecting the two subtrees of T induced by the removal of an edge. Incidentally, in this particular case, the swap tree associated with a best swap edge, exhibits the property of being a MST of $G - e$. Another popular example arises when T is a *single source shortest paths tree* (SPT) rooted at a given node $r \in V$. In this case, the objective function minimized by T is the

sum of the lengths of all the paths emanating from r, and it is not hard to see that a swap tree $T_{e/e'}$ *does not* coincide, in general, with a SPT rooted in r of $G - e$.

Concerning the algorithmic perspective, we observe that, from a network management point of view, it is desirable to know in advance, for each edge in the network, its respective best swap edge, since we can expect that sooner or later each edge will fail. This approach entails at least two advantages: First, as a failure happens, we are prepared to switch immediately to the emergency network; second, and perhaps most important, we can evaluate in advance the *vitality* of an edge, by measuring the degradation of the network functionality (as conveyed by the chosen optimization function) as soon as we switch to the emergency network. An *all-best-swaps* (ABS) problem is therefore the problem of finding a best swap edge–with respect to a given optimization function– *for every* edge in the network. In the past, the ABS problem has been solved both when the network is a MST and a SPT. In the first case, the fastest solution known to date is an $O(m)$ time algorithm [2], while in the second case, an $O(n^2)$ time algorithm has been presented in [9].

However, in several applications, the used spanning tree is neither an MST nor a SPT. Rather, many network architectures look for minimizing the *diameter* of the spanning tree, that is the length of the longest path in the tree between any two nodes, so that the maximum propagation delay in the network will be as low as possible. A tree exhibiting minimum diameter among all possible spanning trees is named a *minimum diameter spanning tree* (*MDST*). A best swap edge for a failing edge of a MDST is a swap edge that keeps the diameter of the swap tree as low as possible among all possible swap trees, and the ABS problem for a MDST is defined accordingly.

In this paper we present an efficient solution precisely for this latter problem. Our approach makes use of some of the techniques presented in [8], in which the authors consider the related problem of computing a best swap edge in a fully dynamic context, where the original MDST evolves due to repeated insertions and deletions of edges in the graph. The solution presented in [8] computes a best swap in $O(n)$ time per update, using $O(m)$ space and preprocessing time. For the edge deletion case, that is of interest for this paper, the authors make use of a topology tree and a 2-dimensional topology tree [3], augmented with some extra information. This requires $O(m)$ time and space for initialization, and allows to compute the length of the diameter of any tree obtained as a consequence of a swap in $O(n)$ time. Hence, the approach in [8] is more general than what is needed for solving the ABS problem, and if we use it for solving the ABS problem, we spend $O(n^2)$ time and $O(m)$ space and preprocessing time. We get these bounds by computing a best swap in $O(n)$ time for each deleted edge. Recently, Alstrup et al. [1] improved the runtime for computing a best swap in an incremental context (i.e., when no deletions are allowed) to $O(\log^2 n)$. In the same paper, the authors claim that it is possible to maintain the diameters of trees in a dynamic forest under link and cut operations in $O(\log n)$ time and $O(n)$ space and preprocessing time. Using this and some of the results contained in [8], the ABS problem can be solved in the following way: we sequentially

remove each edge in the MDST; when an edge e is removed, we initially compute in $O(\sqrt{m})$ time a set of $O(\sqrt{m})$ selected edges, among which a best swap is contained [8]; hence, to compute a best swap, we sequentially insert each of these selected edges, compute in $O(\log n)$ the new diameter and then remove it; finally, we re-insert e and move to the next edge in the MDST. Therefore, the obtained runtime is $O(n\sqrt{m}\log n)$, with $O(m)$ space and preprocessing time.

Our solution solves the ABS problem for a MDST in $O(n\sqrt{m})$ time and $O(m)$ space, thus strictly improving previous bounds for $m = o(n^2)$. This can be done by adapting the well-known *path halving* compression technique [10] for answering in $O(1)$ amortized time (over a total of $\Theta(n\sqrt{m})$ queries) the following query: *Given a rooted tree T and a pair of nodes y and v in T such that v belongs to the subtree of T rooted at y, what is the length of a longest simple undirected path in T, starting at v and staying within the subtree of T rooted at y?* Furthermore, we compare the diameter of $T_{e/e'}$ against that of a real MDST of $G-e$. Somewhat surprisingly, the ratio between them stays within a small constant factor in the worst case, that is 5/2. Therefore, swapping can be viewed as both efficient and effective at the same time.

The paper is organized as follows: in Section 2 we define the problem more precisely and we propose the algorithm for solving it. In Section 3, we present the adaptation of the path halving compression technique which allows to efficiently solve the ABS problem. In Section 4, we compare the diameter of the tree obtained as a consequence of a best swap of a failing edge e as opposed to the diameter of a real new MDST. Finally, in Section 5, we present conclusions and list some open problems.

2 Solving the ABS problem

2.1 Problem statement

For basic definitions of graph terminologies, we refer to [6]. Let $T = (V, E_T)$ be a spanning tree of a 2-edge connected, undirected graph $G = (V, E)$, of n nodes and m edges, and with a nonnegative real length $|e|$ associated with each edge $e \in E$. Let $\mathcal{D}(T) = \langle d_1, d_2, \ldots, d_k \rangle$ denote a *diameter path* of T, that is a path whose length $|\mathcal{D}(T)|$ is maximum in T. We call $|\mathcal{D}(T)|$ the *diameter* of T. For simplicity, we will use the term diameter also for denoting the diameter path, whenever no confusion arises. T is said to be a *minimum diameter spanning tree* (MDST) of G if it has minimum diameter among all the spanning trees of G. If $e \in E_T$ is a tree edge, a *replacement edge* (or *swap edge*) for e is an edge $f \in E \setminus E_T$ such that $T_{e/f} = (V, (E_T \setminus \{e\}) \cup \{f\})$ is a spanning tree of $G - e = (V, E \setminus \{e\})$. Let $\mathcal{R}_e$ denote the set of replacement edges for e. A *best swap edge* for e with respect to the diameter is an edge $e' \in \mathcal{R}_e$ such that

$$|\mathcal{D}(T_{e/e'})| = \min_{f \in \mathcal{R}_e} \{|\mathcal{D}(T_{e/f})|\}.$$

The *all-best-swaps* (ABS) problem for a MDST is the problem of finding a best swap edge with respect to the diameter for every edge $e \in E_T$.

2.2 The algorithm

To solve the ABS problem efficiently, we will exploit relationships among the original MDST T and the replacing ones. Let us denote the length of the path in T between any two nodes v, v' as $d(v, v')$. Let d_c, with $1 \leq c \leq k$, be the *center* of the diameter path, that is the node in $\mathcal{D}(T)$ for which $|d(d_1, d_c) - d(d_c, d_k)|$ is minimum. Let us denote by M this minimum. Notice that, in general, this node is not unique, since it might exist a sequence of edges of length 0 in $\mathcal{D}(T)$, say $(d_i, d_{i+1}), (d_{i+1}, d_{i+2}), \ldots, (d_{i+j-1}, d_{i+j})$, such that $M = |d(d_1, d_h) - d(d_h, d_k)|, h = i, \ldots, i+j$, or it might exist an edge (d_i, d_{i+1}) in $\mathcal{D}(T)$ of positive length and such that $d(d_1, d_i) = d(d_{i+1}, d_k)$. In the former case, to break the tie, we set $d_c = d_{i+j}$, while in the latter case we set $d_c = d_{i+1}$. Let $\widehat{T}$ denote a source directed tree obtained by rooting T in d_c and orienting the edges towards the leaves. Following [8], we maintain a topology tree and a 2-dimensional topology tree [3, 4], augmented with some extra-information, to efficiently retrieve only $O(\sqrt{m})$ *selected edges* among the $O(m)$ replacement edges, whenever an edge e in T is deleted. In fact, among the selected edges, a best swap is contained (for a proof of it, see [8]).

The general outline of our algorithm is the following:

Algorithm ABS(G, T);
Input: A weighted, 2-edge connected graph $G = (V, E)$ and a MDST $T = (V, E_T)$ of G.
Output: $\forall e = (x, y) \in E_T$, a swap edge $e' : |\mathcal{D}(T_{e/e'})| = \min_{f \in \mathcal{R}_e} \{|\mathcal{D}(T_{e/f})|\}$.

`Step 0:` Perform preliminary computations.
`Step 1:` **For** each edge $e \in \widehat{T}$ as considered by any postorder visit **do** {
`Step 2:` Delete e; update the topology and the 2-dim topology tree.
`Step 3:` Compute the set of selected replacement edges $\mathcal{S}_e \subseteq \mathcal{R}_e$.
`Step 4:` For each edge $f \in \mathcal{S}_e$, compute $|\mathcal{D}(T_{e/f})|$ and select a best swap.
`Step 5:` Insert e and update the topology and the 2-dimensional topology tree.}

Step 0 requires $O(m)$ time and space, as we show next. Notice that in Step 1 we consider all the $O(n)$ edges of the tree, in the order they are generated by a postorder tree visit (this order is needed for a correct path halving compression, as is shown in Section 3). Steps 2, 3 and 5 can be accomplished in $O(\sqrt{m})$ time, and $|\mathcal{S}_e| = O(\sqrt{m})$ [8]. Concerning Step 4, we will show that $|\mathcal{D}(T_{e/f})|$ can be computed in $O(1)$ amortized time. This can be done using the *path halving compression* technique that will be presented and analyzed in detail in Section 3. This technique, given a rooted tree $\widehat{T}$ and a pair of nodes u and v in $\widehat{T}$ such that u belongs to the subtree of $\widehat{T}$ rooted at v, allows to find in $O(\log_{(1+Q/n)} n)$ amortized time per query (over a total of Q queries) the length *Findpath*(u, v) of a longest simple undirected path starting from u and staying within the subtree rooted at v. Since for solving the ABS problem we will prove that $Q = \Theta(n\sqrt{m})$, *Findpath*(u, v) can be computed in $O(1)$ amortized time. Thus, Step 4 costs $O(\sqrt{m})$ amortized time, and the global time for solving the ABS problem is $O(n\sqrt{m})$, using $O(m)$ space.

2.3 Preliminary computations

Let $\mathcal{L}_\mathcal{D} = \langle d_1, \ldots, d_{c-1}, d_c \rangle$ and $\mathcal{R}_\mathcal{D} = \langle d_c, d_{c+1}, \ldots, , , d_k \rangle$. W.l.o.g., let us assume $|\mathcal{L}_\mathcal{D}| \geq |\mathcal{R}_\mathcal{D}|$. Let N_l (N_r) denote the subset of nodes of $\widehat{T}$ rooted in d_{c-1} (d_{c+1}), and let N_c denote the nodes of $\widehat{T}$ neither in N_l nor in N_r. Figure 1 depicts this notation.

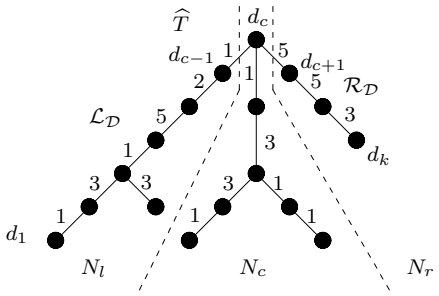

Figure 1: The oriented minimum diameter spanning tree.

We start by marking any node $v \in \widehat{T}$ with its *nearest ancestor* on $\mathcal{D}(T)$, that is the first node of $\mathcal{D}(T)$ we encounter on the path in $\widehat{T}$ from v to d_c. Notice that if v is on the diameter, then it is marked with itself. After, we associate with each node $v \in \widehat{T}$ its distance from d_c, and the lengths $h_1(v)$ and $h_2(v)$, with $h_1(v) \geq h_2(v)$, of the two longest directed paths in $\widehat{T}$ emanating from v and making use of two different subtrees of v (notice that if v is a leaf, then we set $h_1(v) = h_2(v) = 0$, while if there is only one subtree rooted at v, then we set $h_2(v) = 0$). Moreover, we also store with v the node $a(v)$ adjacent to v and belonging to the path of length $h_1(v)$ (notice that if $h_1(v) = 0$, then we set $a(v) = v$). With the root, we also associate a further value, say $h_3(d_c)$, corresponding to the length of a longest path in $\widehat{T}$ starting from d_c and not using d_{c-1} and d_{c+1} (notice that if such a path does not exist, then we set $h_3(d_c) = 0$). After, we associate with each node $d_j \in \mathcal{L}_\mathcal{D}$ ($d_j \in \mathcal{R}_\mathcal{D}$) a further value, say $\lambda(d_j)$, containing the length of a longest path in $\widehat{T}$ starting from d_c and containing neither d_{c+1} (d_{c-1}) nor the edge (d_j, d_{j-1}) ((d_j, d_{j+1})). Note that since d_c belongs to both $\mathcal{L}_\mathcal{D}$ and $\mathcal{R}_\mathcal{D}$, $\lambda(d_c)$ coincides with $h_3(d_c)$. We express $\lambda(d_j)$ recursively as follows:

$$\lambda(d_c) = h_3(d_c)$$
$$\lambda(d_j) = \max(\lambda(d_{j+1}), d(d_c, d_j) + h_2(d_j)) \qquad \text{for } j = c-1, \ldots, 1$$
$$\lambda(d_j) = \max(\lambda(d_{j-1}), d(d_c, d_j) + h_2(d_j)) \qquad \text{for } j = c+1, \ldots, k.$$

Next, we associate with each node $d_j \in \mathcal{L}_\mathcal{D}$ ($d_j \in \mathcal{R}_\mathcal{D}$) a further value, say $\mu(d_j)$, containing the length of a longest path in T starting from d_1 (d_k) and staying within the subtree of $\widehat{T}$ rooted at d_j. We express $\mu(d_j)$ recursively as

follows:

$$\begin{aligned}
&\mu(d_1) = \mu(d_k) = 0 \\
&\mu(d_j) = \max(\mu(d_{j-1}), d(d_1, d_j) + h_2(d_j)) && \text{for } j = 2, \ldots, c-1 \\
&\mu(d_j) = \max(\mu(d_{j+1}), d(d_k, d_j) + h_2(d_j)) && \text{for } j = k-1, \ldots, c+1.
\end{aligned}$$

We also associate with each node d_j on the diameter a further value, say $\rho(d_j)$, containing the nearest node along the diameter of the path stored in $\mu(d_j)$ (this can be done during the computation of $\mu(d_j)$). It is easy to see that all the above computations cost $O(n)$ time.

Finally, we convert G into a graph $G' = (V', E')$ with maximum node degree 3, in the following way [6]: for each node $v \in V$ of degree $\delta(v) > 3$, having nodes $w_1, \ldots, w_{\delta(v)}$ adjacent to it, replace v with nodes $v_1, \ldots, v_{\delta(v)}$; then, add *dashed edges* $(v_i, v_{i+1}), i = 1, \ldots, \delta(v) - 1$, each of length 0, and replace edges $(v, w_i), i = 1, \ldots, \delta(v)$, with edges (v_i, w_i) of corresponding lengths. As a result of this transformation, the graph keeps its original edges, and has an additional $O(m)$ edges of length 0. It is not hard to see that $|V'| \leq 2m$ and $|E'| \leq 3m$, and therefore a MDST T' of G' can be derived from a MDST T of G in $O(m)$ time in the following way: we keep in T' all the edges in T, and for each node $v \in V$ we add the edges $(v_i, v_{i+1}), i = 1, \ldots, \delta(v) - 1$, of length 0. Then, we associate with T' a *topology tree* and a *2-dimensional topology tree* [3, 4], both augmented with some extra-information useful for solving our problem [8]. A topology tree Γ associated with T' is a hierarchical representation of G' based on T', while a 2-dimensional topology tree Γ' associated with Γ is a hierarchical representation of G' based on Γ. The structures of Γ and Γ' are quite involved, and we refer the reader to [3] for an exhaustive description of them. For our scopes, it suffices to mention that Γ and Γ', augmented with some extra-information [8], can be initialized in $O(m)$ time, and allow to find a best swap in a MDST undergoing to edge failures in $O(n)$ time. However, time needed to update Γ and Γ' is just $O(\sqrt{m})$, and since the approach in [8] is more general than what is needed for solving the ABS problem, we will show that this time is enough for computing a best swap in our case.

It is easy to see that to a swapping of an edge in T corresponds the same swap in T', and, vice versa, to a swapping of a non-dashed edge in T' corresponds the same swap in T. Therefore, in the following we will refer to the original spanning tree T, even though our algorithm makes use of the topology tree and the 2-dimensional topology tree associated with T'.

Summarizing, preliminary computations have an overall cost of $O(m)$ time and use $O(m)$ space.

2.4 Computing $|\mathcal{D}(T_{e/f})|$ in $O(1)$ amortized time

In the rest of the paper, two paths will be considered *adjacent* if they share the root d_c only. When the edge $e = (x, y)$ is removed, $\widehat{T}$ is split into two subtrees, say T_x and T_y, containing x and y, respectively, which will be later connected

by means of a replacement edge $f = (u, v)$. As a consequence

$$|\mathcal{D}(T_{e/f})| = \max\{|\mathcal{D}(T_x)|, |\mathcal{D}(T_y)|, |\mathcal{P}_f|\} \quad (1)$$

where $|\mathcal{P}_f|$ is the length of a longest path in $T_{e/f}$ passing through f. Notice that (1) holds for any (possibly not MDST) spanning tree T of G, any edge e in T, and any swap edge f of e. Figure 2 shows the notations used.

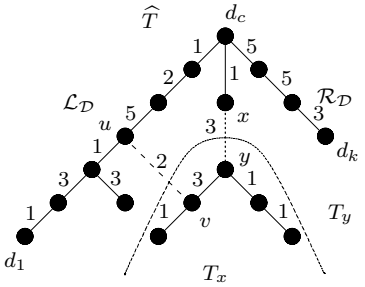

Figure 2: Edge $e = (x, y)$ is replaced by edge $f = (u, v)$.

We now analyze different cases that can arise in solving the ABS problem. For the sake of clarity, we perform a different analysis depending on whether the removed edge is located on the diameter or not.

Case 1: The removed edge is not on the diameter
Assume the edge $e = (x, y)$ is removed, where x is the parent of y in $\widehat{T}$, and $e \notin \mathcal{D}(T)$. In this case, neither $\mathcal{L}_\mathcal{D}$ nor $\mathcal{R}_\mathcal{D}$ are affected. Trivially, $\mathcal{D}(T_x) = \mathcal{D}(T)$. Moreover, $|\mathcal{D}(T_y)| \leq |\mathcal{D}(T)|$, since a path in T_y is a path in T as well. It then remains to compute $|\mathcal{P}_f|$, for any selected replacement edge $f \in \mathcal{S}_e$. Let $f = (u, v)$, where $u \in T_x$ and $v \in T_y$. It is clear that

$$|\mathcal{P}_f| = |\mathcal{L}_u| + |f| + |\mathcal{L}_v|$$

where $\mathcal{L}_u$ is a longest path in T_x starting from u and $\mathcal{L}_v$ is a longest path in T_y starting from v. Since v is a descendant of y in $\widehat{T}$, by using the path halving compression technique $|\mathcal{L}_v| = \mathit{Findpath}(v, y)$ can be computed in $O(1)$ amortized time, as we will see in Section 3, while $|f|$ is clearly available in $O(1)$ time. It remains to compute $|\mathcal{L}_u|$. The following claim is easy to prove:

Lemma 2.1 *At least one of the longest paths in T_x starting from u contains d_c.*

Proof: Suppose, for the sake of contradiction, that none of the longest paths in T_x starting from u contains d_c. Let us restrict our attention to any one of such longest paths, say $\mathcal{P}_u$. We will show that such a path can be modified into another path at least as long as $\mathcal{P}_u$ and passing through d_c, from which the claim will follow. Let w be the node in $\mathcal{P}_u$ nearest to d_c, and let z be the ending node of $\mathcal{P}_u$ other than u. Three cases are possible:

1. $w \in N_l$: let $q \in \mathcal{L}_\mathcal{D}$ be the node on $\mathcal{D}(T)$ nearest to w (if w is on the diameter, then q coincides with w). It is trivial to see that in this case, being $q \neq d_c$ since $w \in N_l$, it must be

$$d(q, z) \leq d(q, d_k)$$

since otherwise $d(d_1, q) + d(q, z) > d(d_1, q) + d(q, d_k) = |\mathcal{D}(T)|$. Being $d(q, z) = d(q, w) + d(w, z)$, it then follows that $\mathcal{P}_u$ can be modified into the path $\mathcal{P}_u{}' = \langle u, \ldots, w, \ldots, q, \ldots, d_k \rangle$ containing d_c and such that

$$|\mathcal{P}_u{}'| = d(u, w) + d(w, q) + d(q, d_k) \geq d(u, w) + d(q, d_k) \geq$$

$$\geq d(u, w) + d(q, z) \geq d(u, w) + d(w, z) = |\mathcal{P}_u|$$

that is a contradiction (see Figure 3a).

2. $w \in N_r$: this case is symmetric to the first one.

3. $w \in N_c$: in this case, it must be clearly $d(w, z) \leq d(w, d_c) + d(d_c, d_1)$, since otherwise $d(d_c, d_1) + d(d_c, w) + d(w, z) > |\mathcal{D}(T)|$. It then follows that $\mathcal{P}_u$ can be modified into the path $\mathcal{P}_u{}' = \langle u, \ldots, w, \ldots, d_c, \ldots, d_1 \rangle$, containing d_c and such that

$$|\mathcal{P}_u{}'| = d(u, w) + d(w, d_c) + d(d_c, d_1) \geq d(u, w) + d(w, z) = |\mathcal{P}_u|$$

that is a contradiction (see Figure 3b).

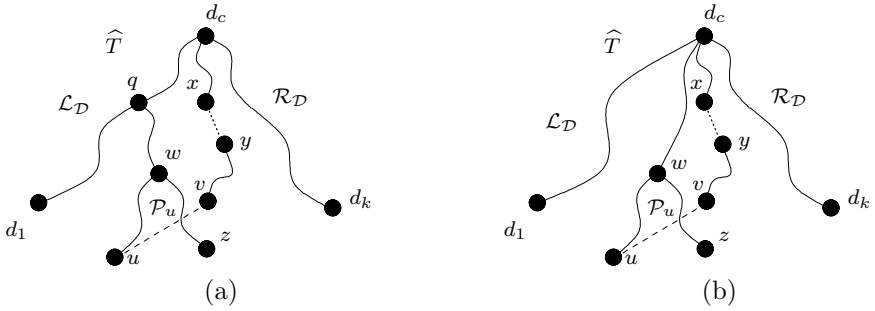

Figure 3: When edge $e = (x, y)$ is removed, there exists a longest path starting from u which passes through d_c (splines denote paths): (a) $w \in N_l$. (b) $w \in N_c$.

□

From the above analysis and from the fact that $\mathcal{L}_\mathcal{D}$ is one of the longest paths emanating from d_c in $\widehat{T}$ and that $\mathcal{R}_\mathcal{D}$ is one of the longest paths emanating from d_c in $\widehat{T}$ which does not make use of d_{c-1} it follows that

$$|\mathcal{L}_u| = \begin{cases} d(d_c, u) + |\mathcal{R}_\mathcal{D}| & \text{if } u \in N_l; \\ d(d_c, u) + |\mathcal{L}_\mathcal{D}| & \text{otherwise.} \end{cases} \quad (2)$$

Therefore, $|\mathcal{L}_u|$ is available in $O(1)$ time. Summarizing, $|\mathcal{P}_f|$ can be computed in $O(1)$ amortized time, and

$$|\mathcal{D}(T_{e/f})| = \max(|\mathcal{D}(T)|, |\mathcal{P}_f|).$$

Once this value is computed for the $O(\sqrt{m})$ selected edges identified by the topology tree and the 2-dimensional topology tree, a best replacement edge is available. Therefore, the case $e \notin \mathcal{D}(T)$ can be managed in $O(\sqrt{m})$ amortized time.

Case 2: The removed edge is on the diameter
We will analyze the case in which $e = (d_i, d_{i-1}) \in \mathcal{L}_\mathcal{D}$ is removed, since the case $e \in \mathcal{R}_\mathcal{D}$ is symmetric. When e is removed, $\widehat{T}$ is split into two subtrees, say T_{d_i} and $T_{d_{i-1}}$, which will be later connected by means of a replacement edge $f = (u, v) \in \mathcal{S}_e$. Equation (1) becomes

$$|\mathcal{D}(T_{e/f})| = \max(|\mathcal{D}(T_{d_i})|, |\mathcal{D}(T_{d_{i-1}})|, |\mathcal{P}_f|).$$

Let us now analyze the value of these three terms.

• $|\mathcal{D}(T_{d_i})|$: We start by proving the following fact:

Lemma 2.2 *At least one of the diameters of T_{d_i} contains $d_k \in \mathcal{R}_\mathcal{D}$.*

Proof: For the sake of contradiction, suppose that none of the diameters of T_{d_i} contains d_k. Let us restrict our attention to any one of such diameters, say $\mathcal{P}$. We will show that in T_{d_i} there is a path containing d_k and at least as long as $\mathcal{P}$, from which the claim will follow. Let w be the node in $\mathcal{P}$ nearest to d_c. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be the two subpaths in which $\mathcal{P}$ splits with respect to w, with ending nodes z_1 and z_2, respectively, and suppose that z_1 precedes z_2 in a preorder traversal of $\widehat{T}$. Figure 4 shows the notations used.

Three cases are possible:

1. $w \in N_l$: this case is similar to the case 1 of the proof of Lemma 2.1. In fact, the modified path there built also contains d_k, apart from d_c.
2. $w \in N_r$: in this case, let $q \in \mathcal{R}_\mathcal{D}$ be the node on $\mathcal{D}(T)$ nearest to w (if w is on the diameter, then q coincides with w). Clearly, any path emanating from q in $\widehat{T}$ is no longer than $d(q, d_k)$. In particular

$$d(q, d_k) \geq d(q, z_2) \geq d(w, z_2).$$

It follows that $\mathcal{P}$ can be modified into the path $\mathcal{P}' = \langle z_1, \ldots, w, \ldots, q, \ldots, d_k \rangle$ containing d_k and such that

$$|\mathcal{P}'| = d(z_1, w) + d(w, q) + d(q, d_k) \geq d(z_1, w) + d(w, z_2) = |\mathcal{P}|$$

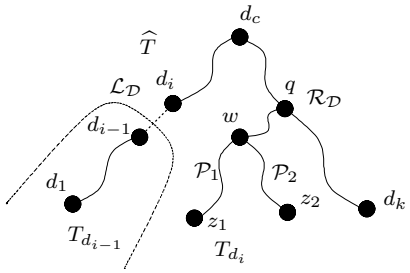

Figure 4: Edge $e = (d_i, d_{i-1})$ is removed: $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$ (splines denote paths).

3. $w \in N_c$: in this case, since z_2 descends from d_c in $\widehat{T}$, it must be clearly $d(w, z_2) \leq d(w, d_c)+d(d_c, d_k)$, since otherwise $d(d_c, d_1)+d(d_c, w)+d(w, z) > |\mathcal{D}(T)|$. It then follows that $\mathcal{P}$ can be modified into the path $\mathcal{P}' = \langle z_1, \ldots, d_c, \ldots, d_k \rangle$ containing d_k and such that

$$|\mathcal{P}'| = d(z_1, w) + d(w, d_c) + d(d_c, d_k) \geq d(z_1, w) + d(w, z_2) = |\mathcal{P}|$$

that is a contradiction.

□

From the above result, it follows that a longest path starting from d_k and not using the edge e just removed can be computed in $O(1)$ time as

$$|\mathcal{D}(T_{d_i})| = \max(\mu(d_{c+1}), \lambda(d_i) + |\mathcal{R}_\mathcal{D}|).$$

• $|\mathcal{D}(T_{d_{i-1}})|$: analogously to the previous case, it can be proved that at least one of the diameters of $T_{d_{i-1}}$ must contain the node d_1. Therefore, it will be

$$|\mathcal{D}(T_{d_{i-1}})| = \mu(d_{i-1})$$

which can be computed in $O(1)$ time.

• $|\mathcal{P}_f|$: let be $f = (u, v)$, where $u \in T_{d_i}$ and $v \in T_{d_{i-1}}$, and $|\mathcal{P}_f| = |\mathcal{L}_u|+|f|+|\mathcal{L}_v|$. $|\mathcal{L}_v| =$ *Findpath*(v, d_{i-1}) can be computed in $O(1)$ amortized time and $|f|$ is available in $O(1)$ time. It remains to analyze $|\mathcal{L}_u|$. Remember that with the node u is associated the nearest node q on the diameter from which it descends. The following three situations are possible for u:

1. $u \in N_l$: in this case, still using the same arguments as for the case 1 of the proof of Lemma 2.1, it can be easily proved that at least one of the longest paths in T_{d_i} starting from u must contain d_c, and then $|\mathcal{L}_u|$ can be obtained in $O(1)$ time as

$$|\mathcal{L}_u| = d(d_c, u) + |\mathcal{R}_\mathcal{D}|.$$

2. $u \in N_r$: In this case, still using the same arguments as for the case 2 of the proof of Lemma 2.2, it can be proved that at least one of the longest paths in T_{d_i} starting from u must contain q. Therefore, it follows that $|\mathcal{L}_u|$ can be obtained in $O(1)$ time as (note that the following is equivalent to compute *Findpath*(u, d_c))

$$|\mathcal{L}_u| = \max\Bigg(d(u,q) + d\Big(q, \rho(d_{c+1})\Big) + \mu(d_{c+1}) - d\Big(d_k, \rho(d_{c+1})\Big), d(u,d_k), d(u,d_c) + \lambda(d_i)\Bigg).$$

3. $u \in N_c$: in this case, it is easy to see that at least one of the longest paths in T_{d_i} starting from u must contain d_c. In fact, for the sake of contradiction, suppose that none of the longest paths in T_{d_i} starting from u contains d_c. Let us restrict our attention to any one of such longest paths, say $\mathcal{P}_u$. Let w be the node in $\mathcal{P}_u$ nearest to d_c, and let z be the ending node of $\mathcal{P}_u$ other than u. Clearly, $d(w,z) \leq d(d_c, d_k)$, and therefore $\mathcal{P}_u$ can be modified into the path $\mathcal{P}_u{}' = \langle u, \ldots, w, \ldots, d_c, \ldots, d_k \rangle$, containing d_c and such that

$$|\mathcal{P}_u{}'| = d(u,w) + d(w,d_c) + d(d_c,d_k) \geq d(u,w) + d(w,z) = |\mathcal{P}_u|$$

that is a contradiction. Given that $\mathcal{L}_u$ contains d_c, it remains to compute the length of a longest path starting from d_c and not containing u, and this can be done by looking at $|\mathcal{R}_\mathcal{D}|$ and at $\lambda(d_i)$ (note that if $\lambda(d_i)$ is exactly the length of a path passing through u, then it follows that $|\mathcal{R}_\mathcal{D}| \geq \lambda(d_i)$). Thus, $|\mathcal{L}_u|$ can be computed in $O(1)$ time as

$$|\mathcal{L}_u| = \max(d(d_c,u) + |\mathcal{R}_\mathcal{D}|, d(d_c,u) + \lambda(d_i)).$$

Summarizing, the case $e \in \mathcal{L}_\mathcal{D}$ can be managed in $O(1)$ amortized time for any of the $O(\sqrt{m})$ selected edges. Since the case $e \in \mathcal{R}_\mathcal{D}$ is symmetric to the previous one, it can be managed with the same amortized runtime. Repeating the above for all the $n-1$ edges of T we therefore have the following:

Theorem 2.1 *The ABS problem for a minimum diameter spanning tree T of a graph G with n nodes and m edges can be solved in $O(n\sqrt{m})$ time, using $O(m)$ space.*

3 Constructing and using compressed paths

In this section we use an adaptation of the well-known *path halving* compression technique to prove that a *Findpath*(v, y) operation can be satisfied in $O(1)$ amortized time, as required to solve efficiently the ABS problem.

We start creating a *virtual forest* $\mathcal{F}$ of trees. Initially, $\mathcal{F}$ is composed of n singletons, one for each node $v \in V$. With each node v in $\mathcal{F}$ the following values are associated: $h_1(v), h_2(v), a(v)$, as defined in Section 2.3, and $up(v)$, which will contain an estimation of the length of a longest path emanating from v and ascending towards d_c in $\widehat{T}$, now considering the edges of the path from d_c to v as directed from v towards d_c. At the beginning, $up(v) = 0$, $\forall v \in \mathcal{F}$. The following instructions manipulate $\mathcal{F}$:

- *Link*(u, v): combine the trees with roots u and v into a single tree rooted in u, adding the edge $e = (u, v)$ of length $|e|$;
- *Eval*(v): Return the length of a longest path starting from v in the tree containing it and apply a suited path halving compression technique.

Note that *Eval*(v) assumes that a pointer to element v is obtained in constant time. The sequence of *Link*() operations in $\mathcal{F}$ is determined by the sequence of edge removals from $\widehat{T}$. Remember that we sequentially consider all the edges $e \in E_T$ in postorder fashion. When the edge $e = (x, y)$ is (temporarily) removed, we perform a sequence of *Link*(y, z_i) in $\mathcal{F}$, where $z_i, i = 1, \ldots, k$ are all the sons of y in $\widehat{T}$. Figure 5 illustrates the situation.

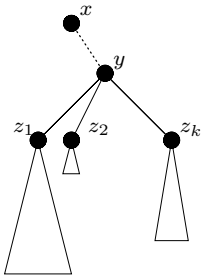

Figure 5: Edge $e = (x, y)$ is deleted and k *Link*() operations are applied.

This means that for any given node $v \in V$, whenever a *Findpath*(v, y) in $\widehat{T}$ occurs, the node y is exactly the root of v in $\mathcal{F}$. In fact, remember that we only ask *Findpath*(v, y) on nodes descending from the currently removed edge. This observation is crucial for the correctness of the method. We implement a *Findpath*(v, y) operation in $\widehat{T}$ by means of an *Eval*(v) operation in $\mathcal{F}$, that examines all the nodes along the path from v to the root y of the tree containing it and compresses such a path. The compression technique used is an adaptation of the *path halving* technique [10], which will guarantee the *associativity* of *Eval*(v). Let us describe how the path halving works. Given a pair of nodes $v_1, v_2 \in V$, with v_1 ancestor of v_2 ($v_1 \prec v_2$) in $\widehat{T}$, we define the following function

$$h(v_1, v_2) = \begin{cases} h_2(v_1) & \text{if } a(v_1) \prec v_2 \text{ or } a(v_1) = v_2; \\ h_1(v_1) & \text{otherwise.} \end{cases}$$

It is easy to see that $h(v_1, v_2)$ can be computed in $O(1)$ time, for any $v_1, v_2 \in V$, after $O(n)$ time of preprocessing of $\widehat{T}$. Assume an *Eval*(v) operation occurs and this is the first time an *Eval*() operation takes place. For the sake of simplicity, let us focus on a path of three nodes $\langle y, u, v\rangle$, with $y \prec u \prec v$ in $\widehat{T}$ and such that $|(u, v)| = \alpha$ and $|(y, u)| = \beta$. We have

$$\mathit{Eval}(v) = \max\Big(h_1(v), up(v), \alpha+h(u,v), \alpha+up(u), \alpha+\beta+h(y,v), \alpha+\beta+up(y)\Big).$$

The compression takes place as part of this operation, and replaces the edge (u, v) with an edge (y, v) of length $\alpha + \beta$ and the label $up(v)$ of v with

$$up(v) = \max\Big(up(v), \alpha + h(u, v), \alpha + up(u)\Big).$$

After the compression we hence have

$$\mathit{Eval}(v) = \max\Big(h_1(v), \max\big(up(v), \alpha + h(u, v), \alpha + up(u)\big),$$

$$\alpha + \beta + h(y, v), \alpha + \beta + up(y)\Big)$$

exactly as before the compression. Therefore, we can conclude that *Eval*(v) compresses paths while correctly maintaining longest path information. Figure 6 illustrates the path halving procedure.

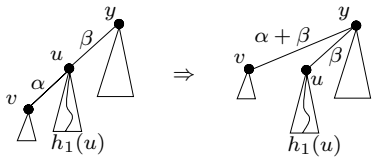

Figure 6: Path halving: Edge (y, v) of length $\alpha + \beta$ replaces edge (u, v) of length α. Note that v continues to take care of the path associated with $h_1(u)$, if needed, since this value is now stored in $up(v)$.

In the general case, let $\langle v_0 = v, v_1, \ldots, v_k = y\rangle$ be the path from v to the root y of the tree containing v in $\mathcal{F}$, having edges $e_i = (v_i, v_{i+1}), i = 0, \ldots, k-1$. We have

$$\mathit{Eval}(v) = \max_{i=1,\ldots,k}\left(h_1(v), up(v), h(v_i, v) + \sum_{j=0}^{i-1}|e_j|, up(v_i) + \sum_{j=0}^{i-1}|e_j|\right). \qquad (3)$$

W.l.o.g., let k be even. The path halving technique makes every other node along the path (except the last and the next to last) point to its grandfather, i.e., replaces the edge $(v_i, v_{i+1}), i = 0, 2, \ldots, k-2$, with the edge (v_i, v_{i+2}) of length $|e_i| + |e_{i+1}|$ and sets

$$up(v_i) = \max\Big(up(v_i), |e_i| + h(v_{i+1}, v_i), |e_i| + up(v_{i+1})\Big), i = 0, 2, \ldots, k-2. \quad (4)$$

Hence, after the halving, the path from v to y is $\langle v_0 = v, v_2, \ldots, v_{k-2}, v_k = y\rangle$, with edges $e'_i = (v_i, v_{i+2}), i = 0, 2, \ldots, k-2$, of length $|e_i|+|e_{i+1}|$. Therefore, after the halving, we have

$$\mathit{Eval}(v) = \max_{i=1,\ldots,k/2}\left(h_1(v), up(v), h(v_{2i}, v) + \sum_{j=0}^{2i-1}|e_j|, up(v_{2i}) + \sum_{j=0}^{2i-1}|e_j|\right)$$

which, from (4), is equivalent to (3). Therefore, it turns out that *Eval*(v) before and after the halving is invariant. Remembering the order the edges are removed, it is clear that the compression of the paths works correctly (i.e., the compression incrementally proceeds towards the upper levels of $\widehat{T}$, according to the edge removals). Therefore, we conclude that *Findpath*(v, y) is equivalent to *Eval*(v).

Since naive linking in $\mathcal{F}$ must be applied to preserve paths of $\widehat{T}$, a sequence of $Q \geq n$ *Eval*(v) queries in $\mathcal{F}$ can be satisfied in $O\left(Q \log_{(1+Q/n)} n\right)$ time [10]. Therefore, to establish that a single query can be satisfied in $O(1)$ amortized time, it remains to prove that $Q = \Omega(n^{1+\epsilon})$ for some constant $\epsilon > 0$. We now show that $Q = \Theta(n\sqrt{m})$, i.e., $\epsilon \geq 1/2$.

Let us distinguish the edges of T in *basic edges* (i.e., edges joining two nodes contained within the same basic cluster [8] and therefore associated with a node at level 0 in the corresponding topology tree) and *spanning edges* (i.e., edges joining two clusters at the same level $j \geq 0$ of the topology tree). If an edge removed from T is a basic edge, then we have $|\mathcal{S}_e| = \Theta(\sqrt{m})$. On the other hand, if an edge removed from T is a spanning edge and joins two clusters corresponding to nodes at level j in the topology tree, $0 \leq j \leq \lceil \log \sqrt{m} \rceil - 1$, then we have $|\mathcal{S}_e| = \Theta(\sqrt{m}/2^j)$. For each edge in $\mathcal{S}_e$ a *Findpath*() query is issued. Hence the minimum overall number of queries is obtained when as many as possible of the $n-1$ failing edges of T are spanning edges at the highest possible level in the topology tree. Since there will be $\Theta(\sqrt{m}/2^{j+1})$ spanning edges for nodes at level j of the topology tree, i.e., a total of $\Theta(\sqrt{m})$ spanning edges, it follows that the minimum number of queries is posed when $\Theta(\sqrt{m})$ edges of T are spanning and the remaining $\Theta(n - \sqrt{m})$ are basic. Therefore, we have

$$Q = \Omega\left((n - \sqrt{m})\sqrt{m} + \sum_{j=0}^{\lceil \log \sqrt{m} \rceil - 1} \frac{\sqrt{m}}{2^{j+1}} \frac{\sqrt{m}}{2^j}\right) = \Omega(n\sqrt{m})$$

and since $Q = O(n\sqrt{m})$, it follows $Q = \Theta(n\sqrt{m})$, which implies $\epsilon \geq 1/2$.

4 Swapping versus recomputing from scratch

In this section, we will show that even though swapping a failing edge in the best possible way does in general not yield a new MDST, it will not give a tree whose diameter differs all that much from a minimum one. Thus, swapping is quick and not too dirty. Let T_{G-e} be a MDST of $G-e$ and let $\sigma_e = |\mathcal{D}(T_{e/e'})|/|\mathcal{D}(T_{G-e})|$, where, as usual, e' is a best swap edge for e.

We start by observing that the MDST T must contain at least one path of length $\mathcal{D}(T)$ which is a shortest path between its two ending nodes [7]. W.l.o.g., let us assume that such a path is $\langle d_1, \ldots, d_k \rangle$, and therefore $\langle d_c, \ldots, d_1 \rangle$ and $\langle d_c, \ldots, d_k \rangle$ are shortest paths. The following holds:

Theorem 4.1 *For any edge $e = (x, y) \in E_T$, we have $\sigma_e \leq 5/2$. The bound is tight.*

Proof: Let $f = (u, v)$ be a replacement edge for e such that $f \in T_{G-e}$ and $|f|$ is minimum. As usual, let $\mathcal{L}_u$ be a longest path in T_x starting from u and $\mathcal{L}_v$ be a longest path in T_y starting from v. We have

$$|\mathcal{D}(T_{e/e'})| \leq |\mathcal{D}(T_{e/f})| \leq \max\Big(|\mathcal{L}_u| + |f| + |\mathcal{L}_v|, |\mathcal{D}(T)|\Big). \tag{5}$$

If there exists at least another edge $f' \in \mathcal{R}_e$ in T_{G-e}, then $\sigma_e \leq 5/2$ trivially follows from the fact that $|\mathcal{D}(T_{G-e})| \geq |f| + |f'| \geq 2|f|$, $|\mathcal{D}(T_{G-e})| \geq |\mathcal{L}_u|, |\mathcal{D}(T_{G-e})| \geq |\mathcal{L}_v|$ and $|\mathcal{D}(T_{G-e})| \geq |\mathcal{D}(T)|$.

Otherwise, let us assume that $f = (u, v)$ is the only edge in $\mathcal{R}_e$ belonging to T_{G-e}. We distinguish two cases: $e \in \mathcal{D}(T)$ and $e \notin \mathcal{D}(T)$.

1. $e = (x, y) \in \mathcal{D}(T)$: W.l.o.g., let $e \in \mathcal{L}_\mathcal{D}$ and let x be the parent of y in $\widehat{T}$. Let $p(v, v')$ denote the length of a shortest path in $G - e$ between any two nodes v, v'. Trivially, $|\mathcal{L}_u| \leq |\mathcal{D}(T_x)|$ and $|\mathcal{L}_v| \leq |\mathcal{D}(T_y)|$. Moreover, in $G - e$, we have that $p(x, d_k) = d(x, d_k) \geq |\mathcal{R}_\mathcal{D}| \geq |\mathcal{D}(T_x)|/2$. Therefore

$$\max\Big(p(u, x), p(u, d_k)\Big) \geq \frac{|\mathcal{D}(T_x)|}{4}.$$

Similarly, we have that $p(d_1, y) = d(d_1, y) \geq |\mathcal{D}(T_y)|/2$, from which

$$\max\Big(p(v, d_1), p(v, y)\Big) \geq \frac{|\mathcal{D}(T_y)|}{4}.$$

Hence, in $G - e$ there exists a shortest path passing through f at least as long as

$$|f| + \frac{|\mathcal{D}(T_x)|}{4} + \frac{|\mathcal{D}(T_y)|}{4}$$

and whilst $|\mathcal{D}(T_x)| \leq |\mathcal{D}(T_{G-e})|$ and $|\mathcal{D}(T_y)| \leq |\mathcal{D}(T_{G-e})|$, (5) can be rewritten as

$$|\mathcal{D}(T_{e/e'})| \leq \max\Bigg(|f| + \frac{|\mathcal{D}(T_x)|}{4} + \frac{|\mathcal{D}(T_y)|}{4} + \frac{3}{4}\Big(|\mathcal{D}(T_x)| + |\mathcal{D}(T_y)|\Big),$$

$$|\mathcal{D}(T)|\Bigg) \leq \max\Big(|\mathcal{D}(T_{G-e})| + \frac{3}{2}\cdot|\mathcal{D}(T_{G-e})|, |\mathcal{D}(T_{G-e})|\Big) \qquad (6)$$

from which $\sigma_e \leq 5/2$ follows. The bound is tight as shown in Figure 7.

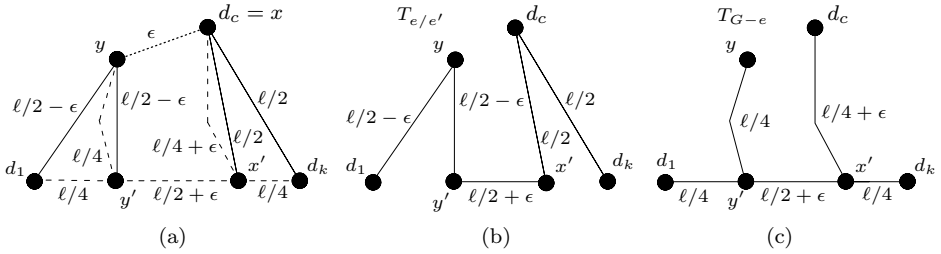

Figure 7: (a) The graph G with the MDST T, having $|\mathcal{D}(T)| = \ell$; non-tree edges are dashed, and edge $e = (x, y)$ is removed from T; (b) $T_{e/e'}$, with the swap edge $e' = (x', y')$, having $|\mathcal{D}(T_{e/e'})| = \frac{5}{2}\ell - \epsilon, \epsilon > 0$; (c) T_{G-e}, having $|\mathcal{D}(T_{G-e})| = \ell + 2\epsilon$.

2. $e = (x, y) \notin \mathcal{D}(T)$: W.l.o.g., let x be the parent of y in $\widehat{T}$. In this case, $\mathcal{D}(T) \in T_x$, and therefore in $G - e$ there exists a shortest path passing through f at least as long as

$$|f| + \frac{|\mathcal{D}(T_x)|}{2}$$

from which (5) can be rewritten as

$$|\mathcal{D}(T_{e/e'})| \leq \max\Big(|f| + \frac{|\mathcal{D}(T_x)|}{2} + \frac{|\mathcal{D}(T_x)|}{2} + |\mathcal{D}(T_y)|, |\mathcal{D}(T)|\Big) \leq$$

$$\leq \max\Big(|\mathcal{D}(T_{G-e})| + \frac{1}{2}\cdot|\mathcal{D}(T_{G-e})| + |\mathcal{D}(T_{G-e})|, |\mathcal{D}(T_{G-e})|\Big) \qquad (7)$$

from which $\sigma_e \leq 5/2$ follows. Also in this case, the bound is tight as shown in Figure 8.

□

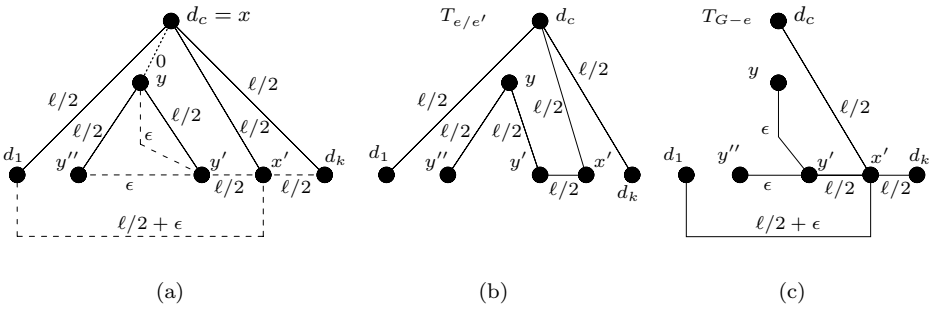

Figure 8: (a) The graph G with the MDST T, having $|\mathcal{D}(T)| = \ell$; non-tree edges are dashed, and edge $e = (d_c, y)$ is removed from T; (b) $T_{e/e'}$, with the swap edge $e' = (x', y')$, having $|\mathcal{D}(T_{e/e'})| = \frac{5}{2}\ell$; (c) T_{G-e}, having $|\mathcal{D}(T_{G-e})| = \ell + 2\epsilon, \epsilon > 0$.

5 Conclusions

In this paper we solved the ABS problem in $O(n\sqrt{m})$ time and $O(m)$ space, thus strictly improving previous bounds for $m = o(n^2)$. We have also shown that the diameter of the tree obtained as a consequence of a best swap is at most 5/2 times the diameter of the real new MDST. It might be interesting to study average values for the above ratio. Another open problem is the case of managing multiple simultaneous edge replacements. Clearly, it also remains to find out whether the algorithm we have proposed is optimal. Finally, the case of transient node failures deserves future study.

Acknowledgments

The authors would like to thank Dagmar Handke for asking the question treated in Section 4, and the referees for the suggestions that helped to improve the presentation of the paper.

References

[1] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Minimizing diameters of dynamic trees. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *24th Int. Coll. on Automata, Languages and Programming (ICALP'97)*, pages 270–280. Lecture Notes in Computer Science Vol. 1256, Springer-Verlag, 1997.

[2] B. Dixon, M. Rauch, and R. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM J. Computing*, 21(6):1184–1192, 1992.

[3] G. Frederickson. Data structures for on-line updating of minimum spanning trees. *SIAM J. Computing*, 14(4):781–798, 1985.

[4] G. Frederickson. Ambivalent data structures for dynamic 2-edge connectivity and k smallest spanning trees. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 632–641, 1991.

[5] M. Grötschel, C. Monma, and M. Stoer. Design of survivable networks. In *Handbooks in OR and MS*, chapter 7, pages 617–672. Elsevier, 1995.

[6] F. Harary. *Graph Theory.* Addison-Wesley, Reading, MA, 1969.

[7] R. Hassin and A. Tamir. On the minimum diameter spanning tree problem. *Information Processing Letters*, 53(2):109–111, 1995.

[8] G. Italiano and R. Ramaswami. Maintaining spanning trees of small diamete. *Algorithmica*, 22(3):275–304, 1998.

[9] E. Nardelli, G. Proietti, and P. Widmayer. How to swap a failing edge of a single source shortest paths tree. In T. Asano, H. Imai, D. Lee, S. Nakano, and T. Tokuyama, editors, *5th International Conference on Computing and Combinatorics (COCOON'99)*, pages 144–153. Lecture Notes in Computer Science Vol. 1627, Springer-Verlag, 1999.

[10] R. Tarjan and J. van Leeuwen. Worst-case analysis of set union algorithms. *J. of the ACM*, 31(2):245–281, 1984.