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

Enrico Nardelli[1,2], Guido Proietti[1,3], and Peter Widmayer[4]

[1] Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila
Via Vetoio, 67010 L'Aquila, Italy
nardelli,proietti@univaq.it.
[2] Ist. di Analisi dei Sistemi e Informatica, CNR, V.le Manzoni 30, 00185 Roma, Italy
[3] On leave to Computer Science Dept., Carnegie Mellon University, 15213 Pittsburgh, PA, supported by the CNR under the fellowship N.215.29
[4] Institut für Theoretische Informatik, ETH Zentrum, 8092 Zürich, Switzerland
widmayer@inf.ethz.ch.
The work of this author was partially supported by grant "Combinatorics and Geometr" of the Swiss National Science Foundation.

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 the new spanning tree. Such an optimal replacement is called the *best swap*. As a natural extension, the *all-best-swaps (ABS) problem* is the problem of finding the 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 $O(m + n)$ space, thus improving previous bounds for $m = o(n^2)$.

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 transient 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. In the extreme, a network might be designed as a spanning tree of some underlying graph of all possible links. A sparse network, however, is less likely to survive a transient edge (or node) failure than a mesh with a multitude of connections that can be used as

* This research was carried out while the first two authors visited the third author within the CHOROCHRONOS TMR program of the European Community.

G. Bilardi et al. (Eds.): ESA'98, LNCS 1461, pp. 55–66, 1998.

detours. Therefore, it is important for sparse networks to also take survivability into account from the very beginning.

On the theoretical side, this gave rise to an interesting family of problems on graphs. In particular, it is important to know in advance how some specific, significant feature of the spanning tree changes as a consequence of a transient edge failure. For example, if the spanning tree has minimum length (i.e., is a minimum spanning tree (MST)), the most immediate feature of interest is the total length of the tree itself, and we are interested in finding the edge whose removal results in the greatest increase of the length of the MST. This problem has been studied (at least implicitly) for a decade, and the fastest solution known to date is an $O(m + n \log n)$ time algorithm [8], where $|V| = n$ and $|E| = m$. Another meaningful class of problems arises when the spanning tree is a single-source shortest path tree (SPT), which is another popular network architecture. In such a case, the problem of finding an edge in the tree whose removal results in the largest increase of the distance between the source node r and a given node s has been solved efficiently by Malik et al. [9], who gave an $O(m+n \log n)$ time algorithm. Recently, Nardelli et al. [10] defined a different parameter for measuring the vitality of an edge along a shortest path, looking for the edge whose removal will result in the worst detour to reach the destination node, and they solved the problem in $O(m + n \log n)$ time.

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. Hence, a *minimum diameter spanning tree* (*MDST*) is a spanning tree having minimum diameter among all the spanning trees. For an MDST subject to transient edge failures in the way just described, it makes sense to temporarily substitute the failing edge with a single edge which makes the diameter of the new spanning tree as low as possible (for practical motivations, see [7]). 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.

The related problem of maintaining a spanning tree of small diameter in a fully dynamic context, where a tree evolves due to repeated insertions and deletions of edges, has been solved in [7]. This problem belongs to the family of problems that is concerned with updating the solution of a graph problem after dynamic changes of the graph; for a recent survey, consult [2]. The approach in [7] is more general than what is needed for solving the ABS problem. Hence, if we use it for solving the ABS problem, we spend $O(n^2)$ time and $O(m+n)$ 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 $O(\log^2 n)$. By using some of the results in [7], their approach can be adapted to solve the ABS problem, and the obtained runtime is $O(n\sqrt{m} \log n)$, with $O(m + n)$ space.

In this paper we solve the ABS problem in $O(n\sqrt{m})$ time and $O(m+n)$ space, thus strictly improving previous bounds for $m = o(n^2)$. This can be done by adapting the well-known *path halving* compression technique [11] 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?

The paper is organized as follows: in Section 2 we define the problem more precisely and formally. Section 3 proposes the algorithm for solving the problem. Finally, in Section 4, we present the adaptation of the path halving compression technique which allows to efficiently solve the ABS problem.

2 Problem Statement

Let $G = (V, E)$ be a biconnected, undirected graph, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges, with a nonnegative real length $|e|$ associated with each edge $e \in E$. Let n and m denote the number of vertices and the number of edges, respectively. A graph $H = (V', E')$ is called a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$. If $V' \equiv V$ then H is called a *spanning subgraph* of G. A connected acyclic spanning subgraph of G is called a *spanning tree* of G.

A *simple path* (or a *path* for short) in $G = (V, E)$ is a subgraph (V', E') with $V' = \{v_1, \ldots, v_k | v_i \neq v_j \text{ for } i \neq j\}$ and $E' = \{(v_i, v_{i+1}) | 1 \leq i < k\}$, also denoted as $\langle v_1, \ldots, v_k \rangle$. Path $\langle v_1, \ldots, v_k \rangle$ is said to go from v_1 to v_k. Because there is exactly one path between any two nodes in a tree, let us denote in this case the length $|\langle v_1, \ldots, v_k \rangle|$ of the path as $d(v_1, v_k)$.

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 the diameter path, whenever no confusion arises. A *minimum diameter spanning tree* of G is a spanning tree of G having minimum diameter. 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 $T_{e,f} = (V, E_T - e + f)$ is a spanning tree of G. Let $\mathcal{R}_e$ denote the set of replacement edges for e. A *best swap* for e is an edge $r(e) \in \mathcal{R}_e$ such that

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

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

3 Solving the ABS Problem

To solve the ABS problem efficiently, we will exploit relationships among the original spanning tree T and the replacing ones. 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. If this node is not unique, we take the node farther from d_1. Let $\widehat{T}$ denote a source directed tree obtained by rooting T in d_c and orienting the edges towards the leaves. Following [7], we maintain an augmented topology tree and an augmented 2-dimensional topology tree [3,4] 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.

3.1 The algorithm

The general outline of our algorithm is the following:

`Step 0:` Perform preliminary computations.
`Step 1:` For each edge $e \in \widehat{T}$ as considered by a postorder visit
`Step 2:` Delete e; update the topology and the 2-dimensional 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 $O(n+m)$ 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 4). Steps 2, 3 and 5 can be accomplished in $O(\sqrt{m})$ time, and $|\mathcal{S}_e| = O(\sqrt{m})$ [7]. 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 4. 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 $O(n + m)$ space.

3.2 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 $|\mathcal{L}_\mathcal{D}| \geq |\mathcal{R}_\mathcal{D}|$. We mark in $O(n)$ time all the nodes in $\widehat{T}$ rooted at d_{c-1}, say N_l, and all the nodes rooted at d_{c+1}, say N_r, with the nearest node on the diameter from which they descend (if a node is on the diameter, then it is marked with itself). N_c will denote the remaining nodes, marked with d_c. Figure 1 depicts this notation.

Then, we associate with each node $v \in \widehat{T}$ its distance from d_c, and the lengths $h_i(v), i = 1, 2$, of the two longest directed paths in $\widehat{T}$ emanating from v and making use of two different subtrees of v, if they exist. For each of these

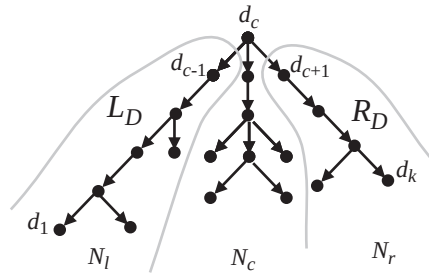

Fig. 1. The oriented minimum spanning tree

paths we also store the nodes $a_i(v), i = 1, 2$, adjacent to v. Let $[h_1(v), a_1(v)]$ and $[h_2(v), a_2(v)]$ be these pairs of values, with $h_1(v) \geq h_2(v)$. 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}. 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:

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

We also associate with each node 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' with maximum vertex degree 3 [6], and we derive from T a spanning tree T' of G' (see [7] for further details). Then, we associate to T' a topology tree and an augmented 2-dimensional topology tree [3,4,7], which can be initialized in $O(m)$ time and space. It is easy to see that an edge swap in T corresponds to an edge swap in T', and vice versa. Therefore, in the following we will continue to refer to the original spanning tree T, even

though our algorithm makes use of the topology tree and the 2-dimensional topology tree of T'.

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

3.3 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, which will be later connected by means of a replacement edge $f=(u,v)$. As a consequence

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

where $|\mathcal{P}_f|$ is the length of a longest path in $T_{e,f}$ passing through f. 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.

3.3.1 The removed edge is not on the diameter

Assume the edge $e=(x,y)$ is removed, where x is closer to d_c than y 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 diameter in T_y is a diameter in T too. 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, while $|f|$ is clearly available in $O(1)$ time. It remains to compute $|\mathcal{L}_u|$. The following claim is easy to prove:

Lemma 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 $q \equiv w$). It is trivial to see that in this case, being $q \not\equiv d_c$ 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.

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. □

¿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

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

and therefore, it follows that $|\mathcal{L}_u|$ 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 augmented topology and 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.

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

Proof. In fact, 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 such diameter can be modified into 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 $\widehat{T}$. Three cases are possible:

1. $w \in N_l$: this case is similar to the case 1 of the proof of Lemma 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 $q \equiv w$). Clearly, any path emanating from q in $\widehat{T}$ is no longer than $d(q, d_k)$. In particular
$$d(q, d_k) \geq d(q, z_1) \geq d(w, z_1)$$
and
$$d(q, d_k) \geq d(q, z_2) \geq d(w, z_2).$$
Since either $\mathcal{P}_1 \cup \langle w, \ldots, q\rangle$ or $\mathcal{P}_2 \cup \langle w, \ldots, q\rangle$ (or both of them) is adjacent to $\langle q, \ldots, d_k\rangle$, it follows that $\mathcal{P}$ can be modified into a no shorter path containing d_k, that is a contradiction.
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 $O(1)$ 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 $|\mathcal{P}_f| = |\mathcal{L}_u| + |f| + |\mathcal{L}_v|$. $|\mathcal{L}_v| = \mathit{Findpath}(v, d_{i-1})$ 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 to 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 point 1 of the proof of Lemma 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 point 2 of the proof of Lemma 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 1. *The ABS problem for a minimum diameter spanning tree T of a graph G with n vertices and m edges can be solved in $O(n\sqrt{m})$ time, using $O(m+n)$ space.* □

4 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, each one associated to a node $v \in V$. With each node v in $\mathcal{F}$ the following values are associated: $[h_i(v), a_i(v)], i = 1, 2$, as defined in Section 3.2 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 $\widehat{T}$. 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 [11], which will guarantee the *associativity* of *Eval*(v). Let us describe how the path halving works. Given a couple of nodes $u, v \in V$, with u ancestor of v ($u \prec v$) in $\widehat{T}$, we define the following function

$$h(u, v) = \begin{cases} h_2(u) \text{ if } a_1(u) \prec v \\ h_1(u) \text{ otherwise.} \end{cases}$$

It is easy to see that $h(u, v)$ can be computed in $O(1)$ time, for any $u, v \in V$, 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

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

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

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

exactly as before the compression. Therefore, we can conclude that $\mathit{Eval}(v)$ compresses paths while correctly maintaining longest path information.

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

$$\mathit{Eval}(v) = \max_{i=1,\ldots,k}\Big(h_1(v), up(v), h(v_i, v) + \sum_{j=0}^{i-1}|e_j|, up(v_i) + \sum_{j=0}^{i-1}|e_j|\Big). \quad (2)$$

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

$$up(v_i) = \max\Big(up(v_i), |e_i| + h(v_{i+1}, v_i), |e_i| + up(v_{i+1})\Big), i = 0, 2, \ldots, k-2. \quad (3)$$

Hence, after the halving, the path between v and 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 (3), is equivalent to (2). Therefore, it turns out that $\mathit{Eval}(v)$ 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 $\mathit{Findpath}(v, y) \equiv \mathit{Eval}(v)$.

Since naive linking in $\mathcal{F}$ must be applied to preserve paths of $\widehat{T}$, a sequence of $Q \geq n$ $\mathit{Eval}(v)$ queries in $\mathcal{F}$ can be satisfied in $O(Q \log_{(1+Q/n)} n)$ time [11]. 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 informally 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 contained inside a basic cluster [7] and therefore associated to a node at level $\ell = 0$ in the corresponding topology tree) and *spanning edges* (i.e., edges joining two clusters at the same level $\ell \geq 0$ 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 ℓ in the topology tree, $0 \leq \ell \leq \lceil \log \sqrt{m} \rceil - 1$, then we have $|\mathcal{S}_e| = \Theta(\sqrt{m}/2^{\ell})$. 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 $\Theta(\sqrt{m}/2^{\ell+1})$ spanning edges for nodes at level ℓ 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

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

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

Acknowledgements – The authors would like to thank anonymous referees for their helpful suggestions.

References

1. S. Alstrup, J. Holm, K. de Lichtenberg and M. Thorup, Minimizing diameters of dynamic trees, *Proc. 24th Int. Coll. on Automata, Languages and Programming (ICALP)*, (1997) 270–280.
2. D. Eppstein, Z. Galil and G.F. Italiano, Dynamic graph algorithms, Tech. Rep. CS96-11, Univ. Ca' Foscari di Venezia (1996).
3. G.N. Frederickson, Data structures for on-line updating of minimum spanning trees, *SIAM J. Computing*, **14** (1985) 781–798.
4. G.N. Frederickson, Ambivalent data structures for dynamic 2-edge connectivity and k smallest spanning trees. *Proc. 32nd IEEE Symp. on Foundations of Computer Science (FOCS)*, (1991) 632–641.
5. M. Grötschel, C.L. Monma and M. Stoer, Design of survivable networks, in: *Handbooks in OR and MS, Vol. 7*, Elsevier (1995) 617–672.
6. F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.
7. G.F. Italiano and R. Ramaswami, Maintaining spanning trees of small diameter, *Proc. 21st Int. Coll. on Automata, Languages and Programming (ICALP)*, (1994) 212–223. A revised version will appear in *Algorithmica*.
8. K. Iwano and N. Katoh, Efficient algorithms for finding the most vital edge of a minimum spanning tree, *Info. Proc. Letters*, **48** (1993) 211–213.
9. K. Malik, A.K. Mittal and S.K. Gupta, The k most vital arcs in the shortest path problem, *Oper. Res. Letters*, **8** (1989) 223–227.
10. E.Nardelli, G.Proietti and P.Widmayer, Finding the detour-critical edge of a shortest path between two nodes, *Info. Proc. Letters*, to appear.
11. R.E. Tarjan and J. van Leeuwen, Worst-case analysis of set union algorithms, *JACM*, **31** (2) (1984) 245–281.