Maintaining a Minimum Spanning Tree under Transient Node Failures*

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

[1] Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila, Via Vetoio, 67010 L'Aquila, Italy. E-mail: {nardelli,proietti}@univaq.it
[2] Istituto di Analisi dei Sistemi e Informatica, Consiglio Nazionale delle Ricerche, Viale Manzoni 30, 00185 Roma, Italy.
[3] Institut für Theoretische Informatik, ETH Zentrum, CLW C 2, Clausiusstrasse 49, 8092 Zürich, Switzerland. E-mail: widmayer@inf.ethz.ch

Abstract. Given a 2-node connected, undirected graph $G = (V, E)$, with n nodes and m edges with real weights, and given a minimum spanning tree (MST) $T = (V, E_T)$ of G, we study the problem of finding, for every node $v \in V$, the MST of $G - v = (V \setminus \{v\}, E \setminus E_v)$, where E_v is the set of edges incident to v in G. We show that this problem can be solved in $\mathcal{O}(\min(m \cdot \alpha(n, n), m + n \log n))$ time and $\mathcal{O}(m)$ space. Our solution improves on the previously known $\mathcal{O}(m \log n)$ time bound.

1 Introduction

Let V be a set of n *sites* that must be interconnected, let E be a set of m potential *links* between the sites, and let $w(e)$ be some real *weight* associated with link e. Let $G = (V, E)$ be the corresponding weighted, undirected graph, and assume that G is 2-node connected (i.e., to disconnect G we have to remove at least 2 nodes). Let $\mathcal{N}_G = (V, E'), E' \subseteq E$ be a connected spanning subgraph of G. We call $\mathcal{N}_G$ a *communication network*, or simply a network, in G. A network is generally built with the aim of minimizing some *cost*, which is computed using some criteria defined over $\mathcal{N}_G$ according to the edge weights. For instance, if the network must be built so that the sum of all the edge weights in the network is minimum, then $\mathcal{N}_G$ is a *minimum (weight) spanning tree* (MST) of G. As another example, if one wants the network to be a spanning tree such that the maximum distance between any two nodes is minimum, then $\mathcal{N}_G$ is a *minimum diameter spanning tree* (MDST) of G.

In addition to minimizing the network cost, we want to focus on another important feature of the network that must be taken into account from the very beginning: Its *reliability* or *survivability*, i.e., the ability of the network to remain operational if individual network components (edges or nodes) fail. In the past

* This work has been partially developed while the second author was visiting the third one, supported by the EU TMR Grant CHOROCHRONOS and by the Swiss National Science Foundation.

few years, several survivability problems have been studied intensely [7]. Clearly, low cost and high reliability of the network are two conflicting objectives. In the extreme, to maintain costs as low as possible, a network might be designed as a spanning tree of the underlying graph. Such a network, however, will not survive even a single edge failure. Even worse, a single node failure in the network might leave each and every node isolated. Therefore, if the network is operational as long as it is connected, some redundancy in the set of edges must be present. As a result, we design the network on two layers: a first layer, where the primary set of links is activated, in such a way that the cost is minimized, and a secondary layer of inactive *replacement links*, a subset of which will switch to active as soon as a network component fails, so that the network will remain connected. We call the activated set of replacement links the set of *swap edges*, given the failed component, and we call the result of swapping the *replacement (swap) network*.

Transient failures of edges and nodes play an important role in reliability considerations: a component that is down is expected to come up again after a little while. Under this assumption, it may be unlikely that there is more than one failure at any given time. Therefore, we study the problem of dealing with the failure of a single edge or node in the network. Since we want to prepare for the transient failure of an arbitrary component, we precompute all individual component failures. We aim at a set of swap edges that ideally minimizes the cost of the replacement network or at least keeps it low. In the extreme, a minimum cost replacement network might entail the use of many swap edges, at a set-up cost per edge in a real communication network. For instance, if the network is a *single source shortest paths tree* (SPT), then the failure of an edge (node) might completely change the shortest paths to all the nodes beyond the failed edge (node), but a change of the network to the new SPT would induce high set-up costs. After the short duration of the transient failure, the switch back to the old SPT would again have high set-up costs, and so it might be desirable to avoid the expensive switch altogether and cope with a worse network for the short time of the failure. Given that we aim at saving set-up cost, we want to choose a small set of replacement edges which will provide a low but not minimum cost replacement network after a failure. In the extreme, for a tree network one could associate with each edge in the network a single swap edge, so that the resulting swap network is the best possible among all the networks that can be obtained by means of a single swap per edge. Similarly, we associate with each node v having $\delta(v)$ adjacent nodes in the network, a set of $\delta(v) - 1$ swap edges that reconnect the network. This is the problem we study in this paper.

In general, even if we choose the best possible swap edges, this *does not* guarantee that the swap network will be the optimum one in the damaged graph, as the previous example with the SPT shows [10]. A similar example is given by the MDST [8,9]. However, for some specific network topologies (i.e., for some specific cost criteria), the swap network is optimal. A very popular network architecture for which this holds is the MST. In fact, it is easy to see that when an edge e fails in a MST, then the MST of $G - e = (V, E \setminus \{e\})$ can be obtained by means of a single replacement edge. Similarly, when a failure happens to a node v having $\delta(v)$ nodes adjacent in the MST, then the MST of $G - v = (V \setminus \{v\}, E \setminus E_v)$, where E_v is the set of edges incident to v in G, can be obtained by means of $\delta(v) - 1$ replacement edges. Thus, reliability in MSTs can be accomplished by using all the best swap edges for both the set of edges and the set of nodes originally in the network.

Not surprisingly then, the problem of computing efficiently all these best swap edges has been studied in the past, in the two settings in which either edge or node failures are considered, starting from different perspectives, though. Let AER (*all edges replacement*) and ANR (*all nodes replacement*) denote the problem of determining the set of swap edges for all the edges and all the nodes in the network, respectively. The fastest solution for solving the AER problem runs in $\mathcal{O}(m)$ time [4]. Note that this positively compares with the fastest dynamic offline algorithm known up to date for maintaining minimum spanning trees, which requires $\mathcal{O}(\log n)$ time per update, and $\mathcal{O}(m)$ space and preprocessing time [5]. In fact, one can always find all the best swaps by considering all the edges of the MST one after the other (and then the algorithm is offline, since we know in advance which edges need to be considered): when the edge e of weight $w(e)$ is considered, we increase its weight to an arbitrary large value, we compute the new MST and we then set back the weight of e to $w(e)$. This will cost a total of $\mathcal{O}(m + n \log n)$ time and $\mathcal{O}(m)$ space.

On the other hand, as far as the ANR problem is concerned, and despite its intrinsic analogy with the AER problem, the best known algorithms are slower, and run either in $\mathcal{O}(m \log n)$ time for $m = o\left(\frac{n^2}{\log n}\right)$ [5], or in $\mathcal{O}(n^2)$ time otherwise [2]. Das and Loui [3] have shown that if the edge weights are sorted in advance, then the ANR can be solved in $\mathcal{O}(m \cdot \alpha(m, n))$ time and space, where here and in the rest of the paper $\alpha(m, n)$ is the functional inverse of Ackermann's function as in [12]. In this way, however, the logarithmic factor is shifted to the sorting of the edge weights, although there seems to be no evidence that edge weights must be sorted for solving the ANR problem.

In this paper, we move one step towards a (potential) linear solution of the ANR problem, by providing an $\mathcal{O}(\min(m \cdot \alpha(n, n), m + n \log n))$ time and $\mathcal{O}(m)$ space algorithm. The problem of finding a linear time algorithm (or, alternatively, a superlinear lower bound) for the ANR problem remains a challenging open problem.

The paper is organized as follows: in Section 2 we define the problem we are dealing with more precisely and formally, and we give some basic definitions that will be used throughout the paper, while in Section 3 we describe the algorithm for solving the problem, and we provide an analysis of both correctness and complexity of the proposed algorithm. Finally, in Section 4, we present conclusions and list some open problems.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph, where V is a set of n nodes and $E \subseteq V \times V$ is a set of m edges, with a real length $w(e)$ associated with each edge

$e \in E$. If multiple edges between nodes are allowed, then the graph is called a *multigraph*. 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 *simple path* (or a *path* for short) in G is a subgraph $P = (V', E')$ with $V' = \{v_1, v_2, \ldots, v_k | v_i \neq v_j \text{ for } i \neq j\}$ and $E' = \{(v_i, v_{i+1}) | 1 \leq i < k\}$, also denoted as $P = \langle v_1, v_2, \ldots, v_k \rangle$. Path P is said to go from v_1 to v_k passing through $v_2, v_3, \ldots, v_{k-1}$. If $v_1 \equiv v_k$, then P is a *cycle*. A graph G is *connected* if, given any two distinct nodes u, v of G, there exists a path from u to v. A graph G is *2-node connected* (*biconnected*, for short) if, given any three distinct nodes u, v, w of G, there exists a path from u to w not passing through v. In other words, a graph G is biconnected if at least 2 of its nodes must be removed to disconnect it. A connected, acyclic spanning subgraph of G is called a *spanning tree* of G. A spanning tree $T = (V, E_T)$ of G is said to be a *minimum weight spanning tree* (MST) of G if the sum of all the edge weights $w(e), e \in E_T$ is minimum for all the spanning trees of G.

Let $T(r)$ denote a MST of a biconnected graph G rooted at an arbitrary node r. Let $v_1, \ldots, v_k$ be the children of node v in $T(r)$, and let v_0 be its parent. Let $G - v = (V \setminus \{v\}, E \setminus E_v)$, where E_v is the set of edges incident to v in G. Note that $G - v$ is connected, since G is biconnected. For any two node-disjoint subtrees T_1 and T_2 of $T(r)$, let $E(T_1, T_2)$ be the set of non-tree edges having one endpoint in T_1 and the other endpoint in T_2. Let us denote by $T(v)$ the subtree of $T(r)$ rooted at v, and by $\overline{T}(v)$ the tree $T(r)$ with $T(v)$ and the edge (v_0, v) removed. Let $\mathcal{H}_v = \{f \in E \setminus E_T | f \in E(T(v_i), T(v_j)), i, j = 1, \ldots, k, i \neq j\}$, referred to in the following as the set of *horizontal edges* of v, and let $\mathcal{V}_v = \{f \in E \setminus E_T | f \in E(T(v_i), \overline{T}(v)), i = 1, \ldots, k\}$, referred to in the following as the set of *vertical edges* of v.

It is easy to see that the MST $T_{\not v}$ of $G - v$ can be computed by making use of the multigraph obtained by *contracting* to a *vertex* each subtree of $T(r)$ created by the removal of v [5]. More precisely, let $\mathcal{W}_v = \{w_0, w_1, \ldots, w_k\}$ be the set of vertices associated with the contraction of the set of subtrees $\{\overline{T}(v), T(v_1), \ldots, T(v_k)\}$. To compute $T_{\not v}$, we first compute the MST $\mathcal{T}_v = (\mathcal{W}_v, \mathcal{R}_v)$ of the multigraph $G_v = (\mathcal{W}_v, \mathcal{H}_v \cup \mathcal{V}_v)$. Then, we set $T_{\not v} = (V \setminus \{v\}, E_T \setminus \{(v_0, v), (v, v_1), \ldots, (v, v_k)\} \cup \mathcal{R}_v)$. Figure 1 illustrates the used notations.

The set $\mathcal{R}_v$ is called the set of *best swap* (or *replacement*) *edges* for v. The *all nodes replacement* (ANR) problem asks for finding $\mathcal{R}_v$ for every node $v \in V$.

3 Solving the ANR Problem

We first give a high-level description of the algorithm, and we then illustrate it in details.

3.1 High-Level Description of the Algorithm

A high-level description of our algorithm is the following. We consider all the non-leaf nodes in $T(r)$ in any arbitrary postorder (if v is a leaf node, then trivially

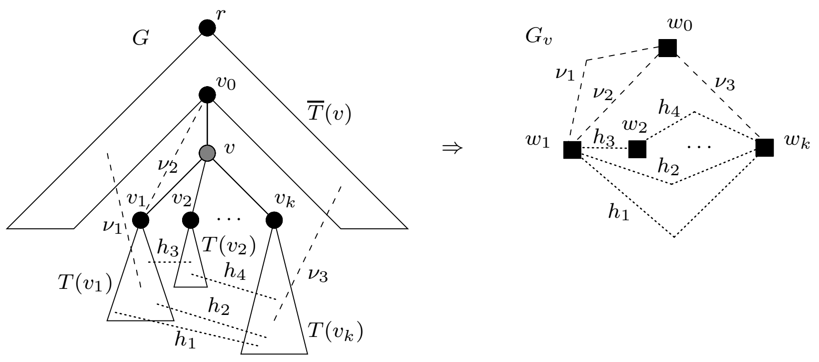

Fig. 1. Node v is removed from G: subtrees $\overline{T}(v), T(v_1), \dots, T(v_k)$ are contracted to vertices $w_0, w_1, \dots, w_k$, respectively, joined by vertical edges (dashed) and horizontal edges (dotted), to form G_v.

$\mathcal{R}_v = \emptyset$). Let us now fix such a node v, having parent v_0 and children $v_1, \dots, v_k$ in $T(r)$, and let $\mathcal{W}_v$, $\mathcal{H}_v$ and $\mathcal{V}_v$ be the corresponding sets of vertices, horizontal and vertical edges, respectively (notice that if $v \equiv r$, then $\mathcal{V}_v = \emptyset$).

Let $f_{v_i} \in E \setminus E_T$ be a *selected vertical edge* for $T(v_i), i = 1, \dots, k$, defined as an edge for which $w(f_{v_i}) = \min\{w(f) | f \in E(T(v_i), \overline{T}(v))\}$, unless $E(T(v_i), \overline{T}(v))$ is empty, in which case f_{v_i} does not exist. For the sake of avoiding technicalities, assume that f_{v_i} is unique (if it exists). Let $\mathcal{V}'_v \subseteq \mathcal{V}_v$ be the set of existing selected vertical edges associated with v. It is easy to see that $\mathcal{R}_v$ corresponds to the set of edges of the MST of the multigraph $H_v = (\mathcal{W}_v, \mathcal{H}_v \cup \mathcal{V}'_v)$. It will turn out that this selection of edges among the vertical edges is vital to get our claimed time complexity for solving the ANR problem.

It is therefore sufficient to compute the MST of H_v, for all the nodes v of $T(r)$ in postorder.

3.2 Computing Efficiently the Selected Vertical Edges

The efficiency problem is the computation of the selected vertical edges $\mathcal{V}'_v$, for all the nodes $v \in V$. It is clearly prohibitive to simply compute $\mathcal{V}'_v$ from scratch, but it is just as well too expensive to attack the problem in a bottom-up fashion, by using mergeable heaps in which each heap contains all the vertical edges associated with a given subtree of $T(r)$. In fact, it can be proved that $\Omega\left(k \log \frac{n}{k}\right)$ time is needed for deleting k elements from a heap of n elements, assuming that insertions, merges and minimum finds are performed in constant time [11]. Then, it is not hard to see that such an approach would require $\mathcal{O}(m \log n)$ time for solving the ANR problem, since $\Theta(m)$ deletions, spread over $\Theta(n)$ heaps containing $\mathcal{O}(n)$ elements at a time, are needed.

To avoid this problem, we adopt a totally different strategy: We find $\mathcal{V}'_v$ by making use of a transformation of the graph G to a multigraph G' containing less than $2m$ edges. After this transformation, we build a *transmuter* [13], representing the set of *fundamental cycles* of G' with respect to $T(r)$, defined as the cycles of G' containing only a single non-tree edge. This will allow us to compute $\mathcal{V}'_v$ in $\mathcal{O}(m \cdot \alpha(m,n))$ time.

We now describe in detail how the transformation works. Let $f = (x,y) \in E \setminus E_T$ denote an arbitrary *non-tree edge* in G. W.l.o.g., we will assume in the following that x precedes y in postorder. Let $nca(x,y)$ denote the *nearest common ancestor* in $T(r)$ of x and y, and let x' and y' denote the children in $T(r)$ of $nca(x,y)$ on the paths (if any) going from $nca(x,y)$ to x and y, respectively (see Figure 2).

Depending on $nca(x,y), x, y, x'$ and y', edge f is either eliminated or is substituted by one or two edges having weight $w(f)$. More precisely, we transform the graph according to the following *substitution rules* (see Figure 2):

(R1): if $nca(x,y) \equiv y$, we substitute f by the edge $f' = (x, x')$;
(R2): if $x' \equiv x$ and $y' \not\equiv y$, we substitute f by the edge $f' = (y, y')$;
(R3): if $y' \equiv y$ and $x' \not\equiv x$, we substitute f by the edge $f' = (x, x')$;
(R4): if $x' \equiv x$ and $y' \equiv y$, f disappears;
(R5): otherwise, f is substituted by the two edges $f' = (x, x')$ and $f'' = (y, y')$.

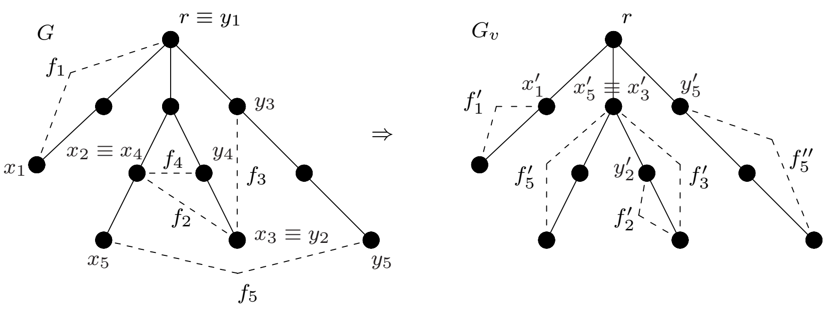

Fig. 2. Substitution of non-tree (dashed) edges $f_i = (x_i, y_i), i = 1, \ldots, 5$, based on $T(r)$ (solid edges), visited in postorder from left to right: edges f_1, f_2, f_3, f_4 and f_5 are substituted according to rules (R1), (R2), (R3), (R4) and (R5), respectively.

The above transformation produces a multigraph $G' = (V, E')$, where E' is the set of original tree edges plus the set of edges substituting the original non-tree edges (*substituting edges*, for short), and therefore $|E'| < 2m$. The decisive property of this transformation is the following:

Lemma 1. *Let $f = (x,y) \in E \setminus E_T$ be a non-tree edge forming a fundamental cycle in G containing a pair of adjacent tree edges $e_v = (v_0, v)$ and $e_{v_i} = (v, v_i)$, where v_0 is the parent of v and v is the parent of v_i in $T(r)$. If x (resp. y) is a descendant of v in $T(r)$, then f is a non-tree edge of minimum weight forming* *a fundamental cycle in G with e_v and e_{v_i}, if and only if $f' = (x, x')$ (resp. $f'' = (y, y')$) is a substituting edge of minimum weight forming a fundamental cycle in G' with e_{v_i}.*

Proof. Let f be a non-tree edge of minimum weight among all the non-tree edges forming a fundamental cycle in G with e_v and e_{v_i}. W.l.o.g., let us assume that x is a descendant of v in $T(r)$ (the case where y is a descendant of v in $T(r)$ can be treated similarly). Since f forms a cycle with e_v and e_{v_i}, it follows that $nca(x, y)$ belongs to the path in $T(r)$ from r to v_0. Hence, the substituting edge f', having weight $w(f) = w(f')$, is such that x' belongs to the path in $T(r)$ from r to v, and therefore f' forms a cycle with e_{v_i}. To show that f' is a substituting edge of minimum weight among all the substituting edges forming a fundamental cycle in G' with e_{v_i}, let us assume that there exists a substituting edge $g' = (u, u')$ forming a cycle with e_{v_i}, and such that $w(g') < w(f')$. Let $g = (u, z)$ be the original non-tree edge substituted by g' (and possibly, by an additional $g'' = (z, z')$). Since g' forms a cycle with e_{v_i}, it follows that u' belongs to the path in $T(r)$ from r to v, and therefore $nca(u, z)$ belongs to the path in $T(r)$ from r to v_0. Hence, g forms a cycle in G with e_v and e_{v_i}, and $w(g) = w(g') < w(f') = w(f)$, a contradiction.

Conversely, w.l.o.g. let f' (the case with f'' can be treated similarly) be a substituting edge of minimum weight among all the substituting edges forming a fundamental cycle in G' with e_{v_i}. The original edge f substituted by f' is such that either $nca(x, y) \equiv v_0$ (according to rules (R1) and (R3)), or $nca(x, y)$ is an ancestor of v_0 in $T(r)$ (according to rule (R5)). In both cases, f forms a fundamental cycle in G with e_v and e_{v_i}, and $w(f) = w(f')$. To show that f is a non-tree edge of minimum weight among all the non-tree edges forming a fundamental cycle in G with e_v and e_{v_i}, let us assume that there exists a non-tree edge $g = (u, z)$ forming a cycle with e_v and e_{v_i}, with u preceding z in postorder and such that $w(g) < w(f)$. Two cases are possible: (1) u is a descendant of v_i in $T(r)$; (2) z is a descendant of v_i in $T(r)$. In the former case, from the fact that $nca(u, z)$ belongs to the path in $T(r)$ from r to v_0, it follows that u' belongs to the path in $T(r)$ from r to v, and therefore g' forms a cycle in G' with e_{v_i}, and $w(g') = w(g) < w(f) = w(f')$, a contradiction. Similarly, in the latter case, it follows that z' belongs to the path in $T(r)$ from r to v, and therefore $g'' = (z, z')$ forms a cycle in G' with e_{v_i}, and $w(g'') = w(g) < w(f) = w(f')$, a contradiction. □

Using the above lemma, we can prove the following result:

Theorem 1. *The selected vertical edges $\mathcal{V}'_v$ for all the non-leaf nodes $v \in V$ can be computed in $\mathcal{O}(m \cdot \alpha(m, n))$ time and $\mathcal{O}(m)$ space.*

Proof. Consider an arbitrary non-leaf node $v \in V$. If $v \equiv r$, then trivially $\mathcal{V}'_v = \emptyset$. Let then v be a non-leaf node other than r, having parent v_0 and children $v_1, \dots, v_k$ in $T(r)$. The multigraph G' can be computed in $\mathcal{O}(m \cdot \alpha(m, n))$ time and $\mathcal{O}(m)$ space: In fact, $nca(x, y)$ can be found in $\mathcal{O}(\alpha(m, n))$ amortized time for each non-tree edge, using $\mathcal{O}(m)$ space [13]. By slightly modifying this algorithm,

x' and y' can be found in $\mathcal{O}(\alpha(m,n))$ amortized time as well: in fact, assuming that $T(r)$ is visited in a fixed postorder, x' can be computed by postponing the union operations appearing in [13] after all the children and the non-tree edges of a given node have been examined, while y' can be computed by reapplying the modified algorithm, but visiting $T(r)$ in reverse postorder. We omit the details due to lack of space.

Moreover, from Lemma 1, it follows that a selected vertical edge $f_{v_i} \in E \setminus E_T$ for the subtree $T(v_i)$, that is a vertical edge of minimum weight containing $e_v = (v_0, v)$ and $e_{v_i} = (v, v_i)$, corresponds to a substituting edge of minimum weight in G' containing e_{v_i}. Henceforth, we can compute efficiently the set of selected vertical edges by means of a transmuter $D[G', T(r)]$ built on G' with respect to $T(r)$. Basically, $D[G', T(r)]$ contains one source node for each tree edge in G', one sink node for each substituting edge in G', and a number of additional nodes. The fundamental property of a transmuter is that there is a path from a given source node to a given sink node if and only if the associated edges form a fundamental cycle in G'. Recall that $D[G', T(r)]$ can be built in $\mathcal{O}(m \cdot \alpha(m,n))$ time [13]. To find a selected vertical edge for the subtree $T(v_i)$, we label a sink node with the weight of the corresponding substituting edge, and we process the transmuter in reverse topological order, labelling each node with the minimum of the labels of its immediate successors. When the process is complete, the source node representing e_{v_i} is associated with a substituting edge of minimum weight forming a cycle with it, which in its turn is associated with an original non-tree edge forming a cycle of minimum weight with e_v and e_{v_i}. Therefore, the edge e_{v_i} remains associated with a selected vertical edge f_{v_i} for the subtree $T(v_i)$, and $\mathcal{V}'_v = \bigcup_{i=1}^{k} \{f_{v_i}\}$. This can be done in $\mathcal{O}(m \cdot \alpha(m,n))$ time and $\mathcal{O}(m)$ space [13], from which the theorem follows. □

3.3 Computing All the Best Swap Edges

Once the vertical edges have been selected, to solve the ANR problem we have to compute the MST of $H_v = (\mathcal{W}_v, \mathcal{H}_v \cup \mathcal{V}'_v)$, whose set of edges corresponds to $\mathcal{R}_v$, for every $v \in V$. This leads to the main result.

Theorem 2. *The ANR problem for a minimum spanning tree of a biconnected graph with n nodes and m edges can be solved in $\mathcal{O}(\min(m \cdot \alpha(n,n), m + n \log n))$ time and $\mathcal{O}(m)$ space.*

Proof. Consider an arbitrary node $v \in V$. If v is a leaf node in $T(r)$, then trivially $\mathcal{R}_v = \emptyset$. Let V' be the set of non-leaf nodes. From Theorem 1, computing $\mathcal{V}'_v$ for every $v \in V'$ costs $\mathcal{O}(m \cdot \alpha(m,n))$ time and $\mathcal{O}(m)$ space. On the other hand, since $\mathcal{H}_v = \{f = (x,y) \in E \setminus E_v | nca(x,y) = v\}$, we can associate $\mathcal{H}_v$ with each node v in $\mathcal{O}(m \cdot \alpha(m,n))$ time and $\mathcal{O}(m)$ space [13].

It remains to analyze the total time needed to compute, for every $v \in V'$, the MST of H_v. Let n_v and m_v denote the number of nodes and the number of edges in H_v, respectively. The MST of H_v can be computed in $\mathcal{O}(m_v \cdot \alpha(m_v, n_v))$ time and $\mathcal{O}(m_v)$ space [1]. Notice that each edge in $\mathcal{H}_v$ is associated with only a single node v, and thus it appears just in a single MST computation, while edges in $\mathcal{V}'_v$ can appear in several MST computations. However, $\sum_{v \in V'} |\mathcal{V}'_v| < n-1$, since we can create at most one selected vertical edge for each tree edge (this also holds for edges leaving the root, with which no selected vertical edges are associated). Notice that this is exactly the reason why vertical edges are selected. In fact, since only $\mathcal{O}(n)$ vertical edges are considered in total, we avoid to consider (in the worst case) $\mathcal{O}(n)$ vertical edges for each of $\mathcal{O}(n)$ failing nodes, i.e., a total of $\mathcal{O}(n^2)$ vertical edges.

It follows that the total time needed to compute the MST of H_v for every $v \in V'$, is

$$\sum_{v \in V'} \mathcal{O}(m_v \cdot \alpha(m_v, n_v)).$$

Since $m_v = \Omega(n_v)$, from the fact that $\alpha(m,n)$ is a monotonically decreasing function in m, it follows that

$$\alpha(m_v, n_v) = \mathcal{O}(\alpha(n_v, n_v)) = \mathcal{O}(\alpha(n, n)).$$

From this and from the fact that $\sum_{v \in V'} m_v = \mathcal{O}(m+n)$, it follows that the total time needed to solve the ANR problem is $\mathcal{O}(m \cdot \alpha(n,n))$, by using $\mathcal{O}(m)$ space.

Notice that the above time bound is worse than $\mathcal{O}(m + n \log n)$ as soon as $m = \omega\left(\frac{n \log n}{\alpha(n,n)}\right)$. Whenever this situation happens, we can use a classic $\mathcal{O}(m_v + n_v \log n_v)$ time and $\mathcal{O}(m_v)$ space algorithm to compute the MST of H_v [6]. This will require $\mathcal{O}(m+n \log n)$ time and $\mathcal{O}(m)$ space to compute $\mathcal{R}_v$ for every $v \in V'$. From this, the thesis follows. □

4 Conclusions

In this paper we have presented an $\mathcal{O}(\min(m \cdot \alpha(n,n), m + n \log n))$ time and $\mathcal{O}(m)$ space algorithm for solving the ANR problem, improving after several years the previously known $\mathcal{O}(m \log n)$ time bound [5].

A natural open problem is how to reduce the time complexity to $\mathcal{O}(m \cdot \alpha(m,n))$, which might be doable by exploiting the relationships among the various MSTs which are computed in postorder. It appears to be much harder to find a linear time algorithm, at least by using our approach which makes use of an MST computation subroutine. Finally, we plan to extend our approach to different network topologies, analyzing the ANR problem for MDSTs and SPTs.

Acknowledgements – The authors would like to thank Gert Stølting Brodal for helpful discussions on the topic.

References

1. B. Chazelle, A minimum spanning tree algorithm with inverse-Ackermann time complexity, TR NECI 99-099, Princeton University, NJ, 1999. 354
2. F. Chin and D. Houck, Algorithms for updating minimal spanning trees, *J. Comput. System Sci.*, **16**(3) (1978) 333–344. 348
3. B. Das and M.C. Loui, Reconstructing a minimum spanning tree after deletion of any node, TR UILU-ENG-95-2241 (ACT-136), University of Illinois at Urbana-Champaign, IL, 1995. 348
4. B. Dixon, M. Rauch and R.E. Tarjan, Verification and sensitivity analysis of minimum spanning trees in linear time, *SIAM J. Comput.*, **21**(6) (1992) 1184–1192. 348
5. D. Eppstein, Offline algorithms for dynamic minimum spanning tree problem, *2nd Workshop on Algorithms and Data Structures (WADS'91)*, Ottawa, Canada, 1991, Vol. 519 of Lecture Notes in Computer Science, Springer-Verlag, 392–399. A revised version appeared in *J. of Algorithms*, **17**(2) (1994) 237–250. 348, 349, 354
6. M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. of the ACM*, **34**(3) (1987) 596–615. 354
7. M. Grötschel, C.L. Monma and M. Stoer, Design of survivable networks, *Handbooks in OR and MS, Vol. 7*, Elsevier (1995) 617–672. 347
8. G.F. Italiano and R. Ramaswami, Maintaining spanning trees of small diameter, *Algorithmica* **22**(3) (1998) 275–304. 347
9. E. Nardelli, G. Proietti and P. Widmayer, Finding all the best swaps of a minimum diameter spanning tree under transient edge failures, *6th European Symp. on Algorithms (ESA'98)*, Venice, Italy, 1998, Vol. 1461 of Lecture Notes in Computer Science, Springer-Verlag, 55–66. 347
10. E. Nardelli, G. Proietti and P. Widmayer, How to swap a failing edge of a single source shortest paths tree, *5th Annual Int. Computing and Combinatorics Conf. (COCOON'99)*, Tokyo, Japan, 1999, Vol. 1627 of Lecture Notes in Computer Science, Springer-Verlag, 144–153. 347
11. D.D. Sleator and R.E. Tarjan, Self-Adjusting Heaps, *SIAM J. Comput.*, **15**(1) (1986) 52–69. 350
12. R.E. Tarjan, Efficiency of a good but not linear set union algorithm, *J. of the ACM*, **22**(2) (1975) 215–225. 348
13. R.E. Tarjan, Applications of path compression on balanced trees, *J. of the ACM*, **26**(4) (1979) 690–715. 351, 352, 353