Nearly linear time minimum spanning tree maintenance for transient node failures

Algorithmica (2004) 40: 119–132
DOI: 10.1007/s00453-004-1099-9

Algorithmica

Nearly Linear Time Minimum Spanning Tree Maintenance for Transient Node Failures[1]

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

Abstract. Given a 2-node connected, real weighted, and undirected graph $G = (V, E)$, with n nodes and m edges, and given a minimum spanning tree (MST) $T = (V, E_T)$ of G, we study the problem of finding, for every node $v \in V$, a set of replacement edges which can be used for constructing an MST of $G - v$ (i.e., the graph G deprived of v and all its incident edges). We show that this problem can be solved on a pointer machine in $\mathcal{O}(m \cdot \alpha(m, n))$ time and $\mathcal{O}(m)$ space, where α is the functional inverse of Ackermann's function. Our solution improves over the previously best known $\mathcal{O}(\min\{m \cdot \alpha(n, n), m + n \log n\})$ time bound, and allows us to close the gap existing with the fastest solution for the edge-removal version of the problem (i.e., that of finding, for every edge $e \in E_T$, a replacement edge which can be used for constructing an MST of $G - e = (V, E\backslash\{e\})$). Our algorithm finds immediate application in maintaining MST-based communication networks undergoing temporary node failures. Moreover, in a distributed environment in which nodes are managed by selfish agents, it can be used to design an efficient, truthful mechanism for building an MST.

Key Words. Graph algorithms, Minimum spanning tree, Transient node failures, Fault tolerance, Algorithmic mechanism design.

1. Introduction.

Let $G = (V, E)$ be a 2-node connected, undirected graph, with n nodes and m edges. Assume that with each edge $e \in E$ a real weight $w(e)$ is associated. Let $T = (V, E_T)$ denote a *minimum spanning tree* (MST) of G, that is, a spanning tree of minimum total edge weight. For any $v \in V$, let $G - v$ denote the subgraph of G induced by the node set $V\backslash\{v\}$ (i.e., the graph G deprived of v and all incident edges). Note that $G - v$ is connected, since G is 2-node connected. In this paper we consider the problem of finding, for every $v \in V$, a set of replacement edges which can be used for constructing an MST of $G - v$.

1.1. *Motivations*.

The MST serves as a basis for many complex network problems. Indeed, assume that V is a set of n *sites* that must be interconnected, through a set E of potential *links* between the sites. Any of these links $e = (u, v)$ has a real value $w(e)$

[1] This work was partially supported by the Swiss National Science Foundation, and by the Research Project GRID.IT, funded by the Italian Ministry of Education, University and Research. A preliminary version of this paper was presented at the 8th European Symposium on Algorithms (ESA 2000), Saarbrücken, Germany, September 5–8, 2000. Volume 1879 of Lecture Notes in Computer Science, Springer-Verlag, Berlin.

[2] Dipartimento di Matematica, Università di Roma "Tor Vergata", Via della Ricerca Scientifica, 00133 Roma, Italy, and Istituto di Analisi dei Sistemi ed Informatica "A. Ruberti", CNR, Viale Manzoni 30, 00185 Roma, Italy. nardelli@mat.uniroma2.it.
[3] Dipartimento di Informatica, Università di L'Aquila, Via Vetoio, 67010 L'Aquila, Italy, and Istituto di Analisi dei Sistemi ed Informatica "A. Ruberti", CNR, Viale Manzoni 30, 00185 Roma, Italy. proietti@di.univaq.it.
[4] Institut für Theoretische Informatik, ETH Zentrum, CLW C 2, Clausiusstrasse 49, 8092 Zürich, Switzerland. widmayer@inf.ethz.ch.

Received December 14, 2003; revised March 9, 2004. Communicated by B. Chazelle.
Online publication June 18, 2004.

associated, representing the cost (in terms of some standard of measurement) for communicating between u and v through e. Then an MST of G is a spanning communication network of *minimum cost*.

In many complex scenarios, computing just a single MST is not sufficient to embed all the system requisites, though. As a classic example, we mention the well-known *fault-tolerance* framework. In this respect the starting point is that—apart from its cost—a communication network must also be *reliable*. Therefore, one should be ready to maintain the connectivity among the sites as soon as any network component (either a node or an edge) fails. Such a maintenance is generally accomplished by activating a *replacement communication network* (RCN) satisfying the same optimization criteria used for building the original network. In particular, for an MST this will result in a *replacement MST* (RMST) of the graph G now deprived of the failed component. Since the failed component is likely to be repaired soon, the RMST is just temporary, and the old, optimal MST will shortly be reactivated. Therefore, under these assumptions, it makes sense to study the problem of dealing with the failure of *every* arbitrary component, to precompute all the individual RMSTs.

Besides the above classical motivation, there is also an emerging and totally different framework in which our problem finds a concrete application, namely the *algorithmic mechanism design*. Indeed, as we discuss in more detail in the paper, the computation of all the RMSTs provides a measure of how the communication network is perturbed by the failure of any of its components, and this reflects the *marginal utility* that each node brings into the MST. Therefore, under the assumption that network components are owned by selfish agents that want to be rewarded to forward messages, we will show that our algorithm can be used to implement a time-efficient mechanism for computing an MST.

1.2. *Related Work*. In the last two decades, several results related to our problem have been obtained, especially for the edge replacement case. In this context it is easy to see that an edge $e \in E_T$ has to be replaced by a minimum weight non-tree edge forming with e a *fundamental cycle* in G (i.e., a cycle containing just a single non-tree edge). Such an edge is named a *replacement edge* for e. The problem of finding all the replacement edges of an MST (i.e., all the RMSTs with respect to edge failures) was originally addressed by Tarjan [20], under the guise of the *sensitivity analysis* of an MST, that is, how much the weight of each individual edge in the MST can be perturbed before the spanning tree is no longer minimal. In his seminal paper, Tarjan solved the problem on a pointer machine in $\mathcal{O}(m \cdot \alpha(m, n))$ time and linear space, where $\alpha(m, n)$ is the functional inverse of Ackermann's function defined in [19]. On the more powerful RAM model, Dixon et al. [6] proposed an optimal deterministic algorithm—for which a tight asymptotic time analysis could not be offered—and a randomized linear time algorithm, while Booth and Westbrook [1] devised a linear time algorithm for the special case in which the graph G is planar.

In a somewhat related scenario, the problem of finding all the RMSTs as a consequence of the failure of each individual node in the graph was originally studied by Chin and Houck [4], who gave an $\mathcal{O}(n^2)$ time algorithm. For not very dense graphs, more precisely for $m = o(n^2/\log n)$, the (more general) offline algorithm for the *dynamic* MST problem given by Eppstein [7] can be used to devise a faster $\mathcal{O}(m \log n)$ time algorithm.

Subsequently, such a bound has been obtained through a different technique from Das and Loui [5], who have also shown that if edge weights are sorted in advance, then the runtime can be lowered to $\mathcal{O}(m \cdot \alpha(m, n))$. In this way, however, the logarithmic factor bottleneck is shifted to the edge weights sorting, although there is no evidence that this operation is crucial for solving the problem. As a matter of fact, this logarithmic factor was removed in [14], where the authors devised an $\mathcal{O}(\min\{m \cdot \alpha(n, n), m + n \log n\})$ time and linear space algorithm. Notice that this algorithm can be suitably modified to produce an optimal linear time algorithm for the planar case [9].

1.3. *Our Result.* In this paper we provide an improved version of the algorithm presented in [14], running in $\mathcal{O}(m \cdot \alpha(m, n))$ time and using $\mathcal{O}(m)$ space. Remarkably, this progress is achieved on a pure pointer machine, without using the direct addressing capability provided by a RAM. Although the asymptotic advancement might appear limited at a first glance, we believe the progress in terms of the problem's knowledge is drastic. Indeed, we obtain the same runtime as for the edge failure case, thus filling the efficiency gap existing between the solution for the edge and the node failure case. The task of finding a linear time algorithm (or, alternatively, a superlinear lower bound) for the two problems remains a challenging open problem.

The paper is organized as follows: In Section 2 we give some definitions and we present the first basic algorithmic observations, while in Section 3 we describe the algorithm for solving the problem, and we provide an analysis of both correctness and complexity. In Section 4 we present an application to a mechanism design optimization problem. Finally, Section 5 contains conclusions and lists 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 weight $w(e)$ 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 $V' = V$, then H is called a *spanning subgraph* of G.

A *path* in G is a subgraph with node set $V' = \{v_1, \ldots, v_k\}$ and edge set $E' = \{(v_i, v_{i+1}) \mid 1 \leq i < k\}$, also denoted as $\langle v_1, v_2, \ldots, v_k \rangle$. Such a path is said to go from v_1 to v_k passing through $v_2, v_3, \ldots, v_{k-1}$. If $v_i \neq v_j$ for $i \neq j$, then the path is called *simple*. If $v_1 = v_k$ and no other nodes are repeated, then the path is called a *cycle*. A graph G is *connected* if, given any two distinct nodes u, v of G, there exists a path going 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 going from u to w not passing through v. In other words, a graph G is biconnected if at least two of its nodes must be removed to disconnect it. A connected, spanning subgraph of G containing no cycles is called a *spanning tree* of G. A spanning tree $T = (V, E_T)$ of G is said to be a *minimum spanning tree* (MST) of G if the sum of all the tree edge weights is minimum among all the spanning trees of G.

Let r denote an arbitrary node in G. In the following the tree T is considered as rooted in r. Let $F = E \backslash E_T$ be the set of non-tree edges of G. For any two node-disjoint subtrees T_1 and T_2 of T, let $F(T_1, T_2)$ be the set of non-tree edges having one endnode in T_1 and the other endnode in T_2. We denote by $T(v)$ the subtree of T rooted at v, and by $\overline{T}(v)$

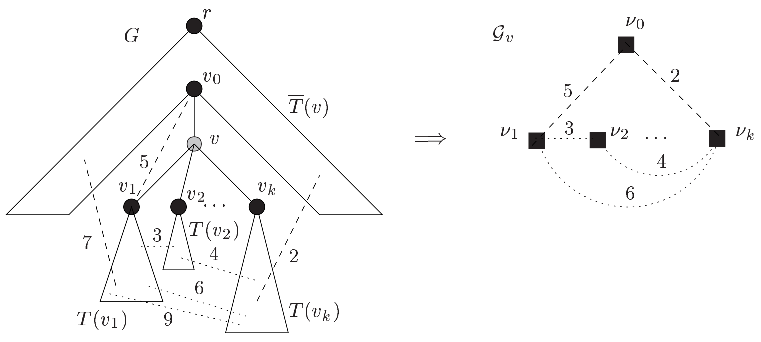

Fig. 1. A node v in G, with the associated set of upwards (dashed) and horizontal (dotted) edges, along with their weights. When node v is removed, subtrees $\overline{T}(v), T(v_1), \ldots, T(v_k)$ are contracted to vertices $\nu_0, \nu_1, \ldots, \nu_k$, respectively, joined by the selected upwards and horizontal edges of v, to form $\mathcal{G}_v$.

the tree T after the removal of $T(v)$. Let v_0 and $v_1, \ldots, v_k$ be the parent and the children of v in T, respectively. Let $\mathcal{H}_v = \{f \in F \mid f \in F(T(v_i), T(v_j)), 1 \leq i, j \leq k, i \neq j\}$, referred to in the following as the set of *horizontal edges* of v, and let $\mathcal{U}_v = \{f \in F \mid f \in F(T(v_i), \overline{T}(v)), 1 \leq i \leq k\}$, referred to in the following as the set of *upwards edges* of v. A *selected horizontal edge* for v with respect to $T(v_i)$ and $T(v_j)$ is an edge (if any) in $F(T(v_i), T(v_j))$ of *minimum* weight. Similarly, a *selected upwards edge* for v with respect to $T(v_i)$ is defined as an edge (if any) in $F(T(v_i), \overline{T}(v))$ of *minimum* weight.

Let $\mathcal{H}'_v \subseteq \mathcal{H}_v$ and $\mathcal{U}'_v \subseteq \mathcal{U}_v$ be the set of selected edges associated with v. It is easy to see that an MST of $G-v$, say T_{G-v}, can be computed through the computation of an MST $\mathcal{T}_v = (\mathcal{V}_v, \mathcal{R}_v)$ of the *contracted graph* $\mathcal{G}_v = (\mathcal{V}_v, \mathcal{H}'_v \cup \mathcal{U}'_v)$, where $\mathcal{V}_v = \{\nu_0, \nu_1, \ldots, \nu_k\}$ is obtained by contracting to a *vertex* each subtree of T created after the removal of v (see Figure 1). Therefore

$$T_{G-v} = (V \backslash \{v\}, E_T \backslash \{\{v_0, v\}, \{v, v_1\}, \ldots, \{v, v_k\}\} \cup \mathcal{R}_v), \tag{1}$$

where $\mathcal{R}_v \subseteq \mathcal{H}'_v \cup \mathcal{U}'_v$ is called the set of *replacement edges* for v.

The All Nodes Temporary Removal problem with input G and T, denoted in the following as ANTR(G, T), is that of finding T_{G-v} (i.e., $\mathcal{R}_v$) for every node $v \in V$.

3. Solving the ANTR(G, T) Problem.

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

3.1. *High-Level Description of the Algorithm.* A high-level description of our algorithm is the following. First, we compute the selected edges for all the non-leaf nodes of T (if v is a leaf node, then trivially $\mathcal{R}_v = \emptyset$). This can be done in $\mathcal{O}(m \cdot \alpha(m, n))$ time by means of a suitable transformation of G as described later, and is the key for the efficiency of our algorithm. Hence, we consider the graph $\mathcal{G}$ defined by the union of all the contracted graphs $\mathcal{G}_v$, one for each non-leaf node v, and we compute in $\mathcal{O}(m \cdot \alpha(m, n))$ time a *minimum spanning forest* $\mathcal{F}$ of $\mathcal{G}$ [3]. In this way, with each contracted graph $\mathcal{G}_v$ a tree in $\mathcal{F}$ remains associated, corresponding to T_{G-v}, and therefore our result follows.

3.2. *Computing the Selected Horizontal Edges*. As sketched in the previous section, the efficiency problem lies in the computation of the selected edges. We start by proving the following:

LEMMA 1. *The selected horizontal edges for all the non-leaf nodes v of T can be computed in $\mathcal{O}(m)$ time and space.*

PROOF. By definition, each horizontal edge $f = (x, y)$ is associated with a unique triplet of nodes in V, say $\langle lca(x, y), v_x, v_y\rangle$, where $lca(x, y)$ denotes the *least common ancestor* (LCA) in T of x and y, and v_x and v_y denote the two children in T of $lca(x, y)$ on the paths (if any) going from $lca(x, y)$ to x and y, respectively. Node $lca(x, y)$ can be obtained in $\mathcal{O}(1)$ amortized time [2]. Concerning nodes v_x and v_y, they can be computed in $\mathcal{O}(1)$ amortized time in the following way [6]: first, we associate with each node $x \in V$ the set of nodes

$$L(x) = \{v \in V \mid \exists f = (x, y) \in F \text{ such that } v = lca(x, y)\};$$

then we perform a depth-first traversal of T, maintaining a stack of the ancestors of the currently visited node, and when a node $x \in V$ is visited, we report, for each node $v \in L(x)$, the node just above v in the stack.

From this, it follows that computing $F(T(v_i), T(v_j))$ for every pair of (non-leaf) siblings v_i and v_j costs $\mathcal{O}(m)$ time, and selecting all the minimums over these sets costs time proportional to the size of all the sets, that is $\mathcal{O}(m)$ time. Since $\mathcal{O}(m)$ space suffices to perform all the above operations, the claim follows. $\square$

3.3. *Computing the Selected Upwards Edges*. The efficiency problem is the computation of the selected upwards edges $\mathcal{U}'_v$, for all the nodes $v \in V$. It is clearly prohibitive simply to compute $\mathcal{U}'_v$ from scratch, but it is also too expensive to attack the problem in a bottom-up fashion, by using mergeable heaps in which each heap contains all the upwards edges associated with a given subtree of T. Indeed, it is known that $\Omega(k \log(p/k))$ time is needed for deleting $k \leq p$ elements from a heap of p elements, assuming that insertion, merge, and find-minimum operations are performed in constant time [18]. Then it is not hard to produce an instance in which such an approach would require $\Omega(m \log n)$ time. More precisely, it suffices to consider the case in which T degenerates to a path, and for each couple of non-adjacent nodes $x, y \in V$ there exists a non-tree edge $f = (x, y)$. Then it is not hard to see that at the kth step of the bottom-up process, $2 < k \leq n - 1$, there are $k - 2$ deletions from a heap containing $\Omega(n)$ elements, and then they cost $\Omega(k \log(n/k))$ time. Therefore, summing up for all the steps, it turns out that $\Omega(m \log n)$ time is needed.

To avoid this problem, we adopt a totally different strategy. More precisely, we find $\mathcal{U}'_v$ by transforming the graph G into a weighted multigraph $G' = (V, E')$ with $E' = E_T \cup F'$, where F' contains less than $2|F|$ edges, and is obtained from F as follows: let $f = \{x, y\} \in F$, and let $\langle lca(x, y), v_x, v_y\rangle$ be defined as above. Edge f is subject to the following *transformation rules* (notice that x and y are interchangeable), depending on the specified conditions:

(i) if $lca(x, y) = y$, f is transformed into edge $f' = \{x, v_x\}$ of weight $w(f)$;

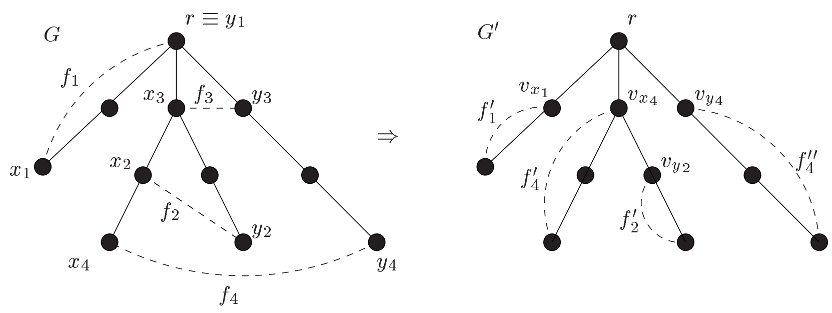

Fig. 2. Transformation of non-tree (dashed) edges $f_i = (x_i, y_i)$, $i = 1, \ldots, 4$, according to the ith transformation rule, based on T (solid edges).

(ii) if $v_x = x$ and $v_y \neq y$, f is transformed into edge $f' = \{y, v_y\}$ of weight $w(f)$;
(iii) if $v_x = x$ and $v_y = y$, f disappears;
(iv) otherwise, f is transformed into edges $f' = \{x, v_x\}$ and $f'' = \{y, v_y\}$, each of weight $w(f)$.

Figure 2 gives an illustration of the above transformation rules (edge weights are omitted for the sake of readability).

The decisive property of the above transformation is the following:

LEMMA 2. *Let $f = (x, y) \in F$ 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, respectively. If x (resp. y) is a descendant of v in T, 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 $f' = (x, v_x)$ (resp. $f'' = (y, v_y)$) is a minimum weight edge of F' forming a fundamental cycle in G' with e_{v_i}.*

PROOF. Let $f = (x, y)$ 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}. Without loss of generality, we assume that x is a descendant of v in T (the case where y is a descendant of v in T can be treated similarly). Since f forms a cycle with e_v and e_{v_i}, it follows that $lca(x, y)$ belongs to the path in T from r to v_0. Hence, the edge $f' = (x, v_x)$, of weight $w(f) = w(f')$, is such that v_x belongs to the path in T from r to v, and therefore f' forms a cycle with e_{v_i}. To show that f' is a minimum weight edge of F' forming a fundamental cycle in G' with e_{v_i}, we assume that there exists an edge $g' = (u, v_u) \in F'$ forming a cycle with e_{v_i}, and, such that $w(g') < w(f')$. Let $g = (u, z)$ be the original non-tree edge transformed into g', and, possibly, into another additional edge $g'' = (z, v_z)$ (see Figure 3). Since g' forms a cycle with e_{v_i}, it follows that v_u belongs to the path in T from r to v, and therefore $lca(u, z)$ belongs to the path in T 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, without loss of generality let $f' = (x, v_x) \in F'$ (the case with $f'' = (y, v_y)$ can be treated similarly) be a minimum weight edge of F' forming a fundamental cycle in G' with e_{v_i}. From the transformation rules, it follows that the original edge f

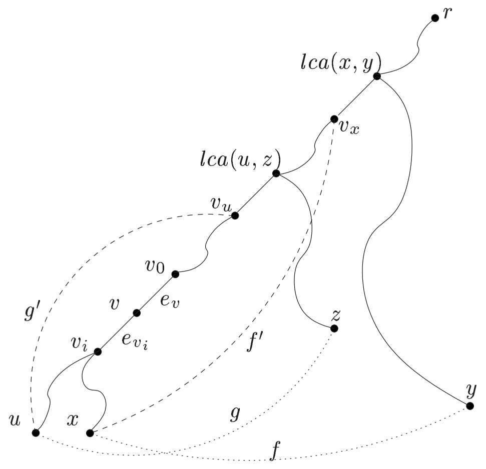

Fig. 3. If $f = (x, y)$ is a non-tree edge of minimum weight among all the non-tree edges forming a fundamental cycle in G with e_v and e_{v_i}, then $f' = (x, v_x)$ must be a non-tree edge of minimum weight among all the non-tree edges forming a fundamental cycle in G' with e_{v_i} (splines denote paths; notice that v_u might coincide with v).

that generated f' is such that either $lca(x, y) = v_0$ or $lca(x, y)$ is an ancestor of v_0 in T. 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}, we assume that there exists a non-tree edge $g = (u, z)$ forming a cycle with e_v and e_{v_i}, and such that $w(g) < w(f)$. Two cases are possible: (1) u is a descendant of v_i in T; (2) z is a descendant of v_i in T. In the former case, from the fact that $lca(u, z)$ belongs to the path in T from r to v_0, it follows that v_u belongs to the path in T from r to v, and therefore $g' = (u, v_u)$ 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 v_z belongs to the path in T from r to v, and therefore $g'' = (z, v_z)$ forms a cycle in G' with e_{v_i}, and $w(g'') = w(g) < w(f) = w(f')$, a contradiction. □

From the above lemma, the following can be proved:

LEMMA 3. *The selected upwards edges for all the non-leaf nodes v of T can be computed in $\mathcal{O}(m \cdot \alpha(m, n))$ time and $\mathcal{O}(m)$ space.*

PROOF. From Lemma 2, computing a selected upwards edge for a node $v \in V$ with respect to any of its children v_i is equivalent to computing a replacement edge for an MST of G' after removing the edge (v, v_i). Hence, the computation of all the selected upwards edges is reduced to an ALL EDGES TEMPORARY REMOVAL problem with input G' and T, namely the AETR(G', T) problem. Since $|E'| < 2m$, this problem can be solved in $\mathcal{O}(m \cdot \alpha(m, n))$ time and $\mathcal{O}(m)$ space [20]. Moreover, G' can be computed in linear time and space [2]. From this, the claim follows. □

3.4. *Finding All the Replacement Edges*. Once the horizontal and the upwards edges have been selected, to solve the ANTR(G, T) problem we have to compute, for every $v \in V$, an MST $\mathcal{T}_v$ of $\mathcal{G}_v$, whose set of edges corresponds to $\mathcal{R}_v$. This leads to the main result:

THEOREM 1. *The* ANTR(G, T) *problem for an MST T of a biconnected, real weighted, undirected graph G with n nodes and m edges can be solved on a pointer machine in $\mathcal{O}(m \cdot \alpha(m, n))$ time and $\mathcal{O}(m)$ space.*

PROOF. From Lemmas 1 and 3, computing $\mathcal{H}'_v$ and $\mathcal{U}'_v$ for every non-leaf node v costs $\mathcal{O}(m \cdot \alpha(m, n))$ time and $\mathcal{O}(m)$ space.

It remains to analyze the total time needed to compute, for every such non-leaf node v, an MST $\mathcal{T}_v$ of $\mathcal{G}_v$. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the graph defined by the set of contracted graphs $\mathcal{G}_v$, one for each non-leaf node $v \in V$. These contracted graphs define the connected components of $\mathcal{G}$. It is easy to see that $\mathcal{G}$ can be built in $\mathcal{O}(m)$ time and space, once that $\mathcal{E}$ is given. Indeed, $|\mathcal{V}| = \mathcal{O}(n)$, since each edge of T generates at most two vertices of $\mathcal{V}$. Moreover, $|\mathcal{E}| = \mathcal{O}(m)$, since each selected horizontal edge is associated with a unique $\mathcal{G}_v$, while there is at most a selected upwards edge for each edge in T. From this, it follows that a minimum spanning forest $\mathcal{F}$ of $\mathcal{G}$ can be computed in $\mathcal{O}(m \cdot \alpha(m, n))$ time and $\mathcal{O}(m)$ space [3]. Since each contracted graph $\mathcal{G}_v$ remains associated with the tree in $\mathcal{F}$ corresponding to T_{G-v}, the claim follows. □

4. An Algorithmic Mechanism Design Application. Interestingly, by solving any network problem which involves the temporary removal, one after the other, of every single component, one naturally obtains a characterization of their *vitality*. Indeed, computing a replacement network provides a measure of the increment in the network cost: the larger the increment, the higher the vitality of the removed component. In standard economic terms, this is also known as the *marginal utility* brought from a component to the network.

In any large network which contains heterogeneous components (e.g., high performance backbone and regional network routers) the above vitality aspect has an immediate economic counterpart. Indeed, each of the network components may be owned by different owners (e.g., an autonomous system), and the incentive for an owner of a component in performing some task (e.g., forwarding a message), naturally, is to get some reward. From the network management point of view, this reward represents the price of the service of forwarding the message. It is economically desirable that each owner declares the true price for the service that her components offer, so as to allocate the overall resources in the best possible way. Nevertheless, there is an incentive for owners to speculate and ask for a higher price, in the hope of getting a higher profit. This leads to economically suboptimal resource allocation and is therefore undesirable. A celebrated game theory result [21] states that when agents are compensated proportionally to their marginal utility, then speculating with high prices does not pay off. Hence, computing (efficiently) these marginal utilities is instrumental to obtain an optimal resource allocation in several network applications.

4.1. *Previous Results*. This interplay between game theory and computational complexity is well known today as *algorithmic mechanism design* for selfish agents [8], [17]. Among others, in their seminal paper [17], Nisan and Ronen addressed the classic *shortest path* problem, which can be formulated as follows. Let a given communication network be modeled by a directed graph $G = (V, E)$, in which two distinguished nodes s and t, called respectively the *source* and the *destination* node, want to establish a communication. Each edge $e = (x, y) \in E$ is owned by an agent A_e, which holds private information $t(e)$ associated with e, named *type*. This value depends on various factors (e.g., bandwidth, reliability, etc.) and expresses the agent's *true* cost (in terms of some common currency) for receiving or sending a message through that edge. Each agent, whenever called for using her edge, will report a (not necessarily true) cost $r(e)$. This value represents her *strategy*. Indeed, on the basis of all the reported costs, a solution (i.e., a path between s and t) will be computed, and if a given edge e lies on this path, then the corresponding owner will receive a *payment* $p(e)$, thus realizing a *utility* $u(e) = p(e) - t(e)$. Since the payments are known in advance to the agents as a function of the computed solution, it is clear that the agents, depending on their strategies, can negatively influence the system, by leading it to compute a suboptimal solution (with respect to their true costs). Therefore, the system-wide goal is to induce, through appropriate payments, the agents to cooperate in computing an optimal output, by revealing their true costs. This combination of output computation and definition of payments is usually referred to as a *mechanism*. We call a strategy *dominant* for an agent if it maximizes the agent's utility, whatever the strategies adopted by the other agents. Then a mechanism is *truthful* if for the agents it is a dominant strategy to report their true costs.

As far as the shortest path problem is concerned, if we denote by $d_G(s, t)$ the length of a shortest path in G between s and t according to the reported edge costs, then the following payment function ensures that for the agents it is a dominant strategy to report their true costs [17]. In this way an authentic shortest path in G between s and t will be selected:

$$(2) \quad p(e) = \begin{cases} 0 & \text{if } e \text{ is not on the shortest path;} \\ d_{G-e}(s, t) - (d_G(s, t) - r(e)) & \text{otherwise.} \end{cases}$$

Indeed, it can be proved that the above payments can be used to implement a so-called *Vickrey–Clarke–Groves mechanism* (*VCG-mechanism*) [10], which enjoys the fundamental property of being truthful. With respect to our simplified framework, a VCG-mechanism can be defined as follows (for an extensive treatment of the subject, the interested reader is referred to [17]):

DEFINITION 1 (VCG-Mechanism). Let $G = (V, E)$ be a graph in which each edge $e \in E$ is owned by an agent A_e, which holds a private type $t(e)$ and reports a public weight $r(e)$ for e. Let $\pi = \langle G, \mathcal{S}, \varphi \rangle$ be a maximization (minimization) problem on G, with $\mathcal{S} \subseteq 2^E$ denoting the set of feasible solutions, and φ denoting the objective function, which is subject to the restriction that, for $E' \in \mathcal{S}$, the following holds:

$$(3) \qquad \varphi(E') = \sum_{e \in E'} r(e).$$

Then a *VCG-mechanism* for π is a pair $\langle \texttt{Alg}, \mathcal{P}\rangle$, where:

1. $\texttt{Alg}$ is an algorithm which finds a feasible solution maximizing (minimizing) φ:
2. $\mathcal{P}$ is the set of payments provided to the agents, where the payment $p(e)$ for the agent A_e has the following form: let $h(t^{-e})$ be an arbitrary function not depending on the type $t(e)$ associated with $e \in E$, and let E' be the solution reported from $\texttt{Alg}$; then

$$p(e) = \begin{cases} 0 & \text{if} \quad e \notin E'; \\ h(t^{-e}) - (\varphi(E') - r(e)) & \text{otherwise.} \end{cases} \tag{4}$$

It is not hard to see that the payments (2) for the shortest path problem fit the above definition.

Based on that, the algorithmic question posed in [17] was the following: How fast can the payment functions (2) be computed? The authors of [17] conjectured this can be done in $\mathcal{O}(m \log n)$ time. Unfortunately, this is not the case, since for $m = \mathcal{O}(n\sqrt{n})$, a lower bound of $\Omega(m\sqrt{n})$ time holds [11]. On the other hand, for undirected graphs, there exists an $\mathcal{O}(m + n \log n)$ time algorithm for a pointer machine [12], and an $\mathcal{O}(m \cdot \alpha(m, n))$ time algorithm for a RAM [15], respectively. Notice that both these papers were motivated by the problem of finding the *most vital edge* of a shortest path. This is not merely a coincidence, as already observed before, since the vitality of an edge reflects the marginal utility it brings into the solution, which is exactly what a VCG-mechanism aims to capture.

For another popular network topology, that is the MST, the situation evolved similarly. Indeed, as pointed out in [17], the $\mathcal{O}(m \cdot \alpha(m, n))$ time and linear space sensitivity analysis algorithm by Tarjan [20] (which computes all the replacement edges of an MST), can be used to implement a truthful mechanism for designing an MST in a communication network in which edges are owned by selfish agents.

Since Definition 1 can be easily extended to the case in which agents control multiple edges in the network, a natural set of problems arises when agents sit on nodes and control a subset of edges incident to that node. In this respect, a first result was obtained in [16], where an $O(m + n \log n)$ time algorithm was presented for the problem of finding a *most vital node* of a shortest path (i.e., a node whose removal induces a longest replacement shortest path between s and t). As the authors pointed out, such an algorithm can be used to implement a truthful mechanism for solving the shortest path problem in a communication network in which nodes are owned by selfish agents. However, what about the problem of finding an MST in such a scenario? In the following we provide both positive and negative results along this direction.

4.2. *Applying Our Algorithm.* Assume then that each node v of the graph is controlled by one or more independent agents. Each of these agents owns (in an exclusive way) a non-empty subset of the edges incident to that specific node. It is worth noticing that this scenario models a realistic situation in which, for instance, each node represents a given geographical region, containing a certain number of servers (each belonging to a different owner). Each server is connected to other nodes by a set of links that can be used to receive and forward messages. Then, in the interest of a best possible overall resource allocation, the system-wide goal is to design a routing protocol in which messages are exchanged through a minimum-cost network, i.e., an MST.

By extending our adopted notation, we have that the above problem can be formalized as follows. Let V be a set of nodes and let E be a set of edges interconnecting all these nodes. Let $\mathcal{A}_v = \{A_v^1, \dots, A_v^k\}$ be the set of agents associated with $v \in V$, and let $E_v^i \subset E$ denote the set of edges incident to v which are owned by agent A_v^i. Agent A_v^i associates with edge $e \in E_v^i$ a type $t(e) \in \mathbb{R}^+$, and reports a cost $r(e) \in \mathbb{R}^+$. Let $G = (V, E)$ be the positively weighted graph obtained by associating with each edge the cost reported by the agent owning it. In the following we assume that G is connected, and then let $T = (V, E_T)$ be an MST of G. If an edge $e \in E_v^i$ belongs to T, then agent A_v^i will receive a payment $p(e) \in \mathbb{R}^+$ for its use, while otherwise $p(e) = 0$. Hence, the global payment of A_v^i will be

$$p_v^i = p_v^i(E_v^i, E_T) = \sum_{e \in E_v^i \cap E_T} p(e), \tag{5}$$

while her utility will be

$$u_v^i = u_v^i(E_v^i, E_T) = \sum_{e \in E_v^i \cap E_T} u(e) = \sum_{e \in E_v^i \cap E_T} [p(e) - t(e)]. \tag{6}$$

Now, let $G^* = (V, E)$ be the weighted graph obtained by associating with each edge e the corresponding type of the agent owning it, and let $T^* = (V, E_T^*)$ be an MST of G^*. In general, depending on the agents' declarations, we have that T and T^* may differ. Nevertheless, the system-wide goal is to obtain T^*, i.e., a true MST, and this is potentially in contrast with the strategies of the selfish agents. To solve the question, once again we have then to design a truthful mechanism.

First, we show a positive result, i.e., the existence of a truthful mechanism, which can be implemented efficiently through our algorithm. Let $G - E_v^i$ denote the graph G deprived of the edges in E_v^i (notice that $G - E_v^i$ might not be connected). Let $T_{G-E_v^i}$ be a minimum spanning forest of $G - E_v^i$, and let $\varphi(T)$ and $\varphi(T_{G-E_v^i})$ denote the weights of T and $T_{G-E_v^i}$, respectively. Finally, let $\varphi(T|_{E_v^i \leftarrow 0})$ denote the weight of T once the weights of the edges in E_v^i have been set to zero. The following payment function is a natural generalization of (4), and ensures that a dominant strategy for the agents is to report their true costs, thus guaranteeing that an optimal solution (i.e., a true MST) will be selected:

$$p_v^i = \begin{cases} 0 & \text{if} \quad E_v^i \cap E_T = \emptyset; \\ \varphi(T_{G-E_v^i}) - \varphi(T|_{E_v^i \leftarrow 0}) & \text{otherwise.} \end{cases} \tag{7}$$

Indeed, the payment of any agent is a linear function of the reported costs of all the other agents, and depends neither on her valuations nor on a function of her types. Therefore, this payments define a VCG-mechanism. Moreover, we can prove the following:

THEOREM 2. *The payments functions* (7) *for all the agents can be computed on a pointer machine in $\mathcal{O}(m \cdot \alpha(m, n))$ time and $\mathcal{O}(m)$ space.*

PROOF. It suffices to show that our algorithm from Section 3 can be adapted in order to compute $T_{G-E_v^i}$ instead of T_{G-v}, for every agent and every node $v \in V$.

First, we modify the definition of the contracted graph $\mathcal{G}_v$. More precisely, for an agent A_v^i, let $\mathcal{G}_v^i$ denote the contracted graph obtained as follows:

1. The vertex set of $\mathcal{G}_v^i$ results from the contraction of each subtree of T created after the removal of E_v^i. If $\mathcal{G}_v^i$ contains only a single vertex (i.e., $E_v^i \cap E_T = \emptyset$), then set $p_v^i = 0$ and discard it.
2. The edge set of $\mathcal{G}_v^i$ consists of the subsets of the selected upwards and horizontal edges of $\mathcal{G}_v$ whose endnodes do not belong to the same contracted vertex of $\mathcal{G}_v^i$ (notice that some of the selected horizontal and upwards edges computed as described in Section 3 might now be absorbed during the contraction). Moreover, let E_v^{-i} be the set of edges incident to v but not contained in E_v^i. Then we also add to the edge set of $\mathcal{G}_v^i$ the subset of edges of E_v^{-i} whose endnodes do not belong to the same contracted vertex.

In this way it is easy to see that the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ defined by this set of modified contracted graphs, can still be built in $\mathcal{O}(m)$ time and space. Moreover, the vertex set of each contracted graph $\mathcal{G}_v^i$ in $\mathcal{G}$ has size equal to $|E_v^i \cap E_T|+1$, and therefore $|\mathcal{V}| \leq 2(n-1)$. Concerning the edge set, notice that each selected upwards and horizontal edge of $\mathcal{G}_v$ appears at most twice in $\mathcal{G}$. Indeed, if the two tree edges incident to v which form a fundamental cycle with a given selected edge belong to the same agent, then the selected edge will be absorbed, while otherwise if the two tree edges belong to different agents, then the selected edge will be duplicated. Finally, each edge of E_v^{-i} which is added to $\mathcal{G}_v^i$ appears at most twice in $\mathcal{G}$ (once for each respective endnode). Thus, $|\mathcal{E}| = \mathcal{O}(m)$. From this, it follows that a minimum spanning forest $\mathcal{F}$ of $\mathcal{G}$ can be computed in $\mathcal{O}(m \cdot \alpha(m, n))$ time and $\mathcal{O}(m)$ space [3]. Since each $\mathcal{G}_v^i$ remains associated with the subforest in $\mathcal{F}$ corresponding to $T_{G-E_v^i}$, the claim follows. □

4.3. *Voluntary Participation*. Unfortunately, the above mechanism is *dictatorial*, in the sense that there might be cases in which an agent is forced to enter into the solution (or, putting it differently, an agent is instrumental to the solution). More precisely, we have that this situation occurs whenever an agent in G exists such that the removal of all her edges disconnects G. In this case any spanning tree of G must include at least one edge of such an agent. This implies an interesting consequence: We cannot guarantee the *voluntary participation* of this agent (i.e., we cannot provide a truthful mechanism such that her utility is always non-negative). Indeed, the following can be proved:

THEOREM 3. *If an agent in G exists such that the removal of all her edges disconnects G, then no truthful mechanism exists for computing an MST of G that guarantees a non-negative utility for that agent.*

PROOF. Let A_z^j be an agent such that $G - E_z^j$ is not connected. For the sake of simplicity, assume that E_z^j consists of a single edge $e_z \in E_T$. By contradiction, assume the claim is not true. From our assumptions, the only truthfully dominant strategy mechanisms for solving the problem belong to the VCG family [13]. Then, from (4), the payment function for A_z^j must be

$$p_z^j = h(t^{-e_z}) - \sum_{e \in E_T \setminus \{e_z\}} r(e).$$

Therefore, to get a non-negative utility for A_z^j, it must be

$$u_z^j = p_z^j - t(e_z) = h(t^{-e_z}) - \sum_{e \in E_T \setminus \{e_z\}} r(e) - t(e_z) \geq 0.$$

Since the mechanism is truthful, we have that $r(\cdot)$ coincides with $t(\cdot)$, and then the above is equivalent to

$$h(t^{-e_z}) \geq \sum_{e \in E_T \setminus \{e_z\}} t(e) + t(e_z). \tag{8}$$

Notice that the above condition must be satisfied as long as e_z participates in the solution, which is always the case. This means that (8) must hold true even if $t(e_z)$ grows arbitrarily. Thus, for any $h(t^{-e_z})$ and any $\varepsilon > 0$, it suffices to have

$$t(e_z) \geq h(t^{-e_z}) - \sum_{e \in E_T \setminus \{e_z\}} t(e) + \varepsilon$$

to have a contradiction. □

5. Conclusions and Future Work. In this paper we have presented an $\mathcal{O}(m \cdot \alpha(m, n))$ time and $\mathcal{O}(m)$ space algorithm for solving on a pointer machine the all nodes replacement problem ANTR(G, T), where G is a 2-node-connected, real weighted graph, and T is an MST of G. This algorithm finds application in managing temporary node failures in MST-based communication networks, and provides an efficient solution to an interesting algorithmic mechanism design problem.

It is worth noticing that we have obtained the same runtime as for the edge-version of the problem, namely the AETR(G, T). Since this problem can be solved on a RAM model in optimal time, although the corresponding tight asymptotic bound is still not known [6], the problem of devising (possibly by modifying appropriately the algorithm presented in [6]) an optimal algorithm for solving the ANTR(G, T) problem on a RAM model remains open. Moreover, for both versions of the problem, it would be interesting to address the question of designing an optimal algorithm on a pointer machine. This seems to be doable by exploiting the results contained in [6] and [2]. We conjecture that the two problems have the same time complexity.

From a different perspective, a natural goal to be pursued is to extend the class of graphs (at the moment restricted to planar graphs [9]) for which a linear bound for the two problems holds. Moreover, in a broader scenario, we mention the problem of studying ANTR and AETR problems for spanning subgraphs of G other than an MST.

Acknowledgments. The authors thank Giovanna Melideo for inspiring discussions on the topic, and the anonymous referees for their very helpful comments.

References

[1] H. Booth and J. Westbrook, A linear algorithm for analysis of minimum spanning and shortest-path trees of planar graphs, *Algorithmica*, **11** (1994), 341–352.

[2] A.L. Buchsbaum, H. Kaplan, A. Rogers, and J. Westbrook, Linear-time pointer-machine algorithms for least common ancestors, MST verification, and dominators, *Proc. of the* 30*th Annual ACM Symposium on Theory of Computing* (*STOC* 1998), pp. 279–288.

[3] B. Chazelle, A minimum spanning tree algorithm with inverse-Ackermann time complexity, *J. Assoc. Comput. Mach.*, **47**(6) (2000), 1028–1047.

[4] F. Chin and D. Houck, Algorithms for updating minimal spanning trees, *J. Comput. System Sci.*, **16**(3) (1978), 333–344.

[5] B. Das and M.C. Loui, Reconstructing a minimum spanning tree after deletion of any node, *Algorithmica* **31** (2001), 530–547. Also available as TR UILU-ENG-95-2241 (ACT-136), University of Illinois at Urbana-Champaign, IL, 1995.

[6] 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.

[7] D. Eppstein, Offline algorithms for dynamic minimum spanning tree problems, *J. Algorithms*, **17**(2) (1994), 237–250.

[8] J. Feigenbaum and S. Shenker, Incentives and Internet computation, *ACM SIGACT News*, **33**(4) (2002), 37–54.

[9] C. Gaibisso, G. Proietti, and R.B. Tan, Optimal MST maintenance for transient deletion of every node in planar graphs, *Proc. of the* 9*th Annual International Computing and Combinatorics Conference* (*COCOON* '03), Vol. 2697 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 404–414.

[10] T. Groves, Incentives in teams, *Econometrica*, **41**(4) (1973), 617–631.

[11] J. Hershberger, S. Suri, and A.M. Bhosle, On the difficulty of some shortest path problems, *Proc. of the* 20*th Symposium on Theoretical Aspects of Computer Science* (*STACS* '03), Vol. 2607 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 343–354.

[12] K. Malik, A.K. Mittal, and S.K. Gupta, The k most vital arcs in the shortest path problem, *Oper. Res. Lett.*, **8** (1989), 223–227.

[13] C. Montet and D. Serra, *Game Theory & Economics*, Palgrave MacMilliam, Houndmills, Hampshire, 2003.

[14] E. Nardelli, G. Proietti, and P. Widmayer, Maintaining a minimum spanning tree under transient node failures, *Proc. of the* 8*th European Symposium on Algorithms* (*ESA* 2000), Vol. 1879 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 346–355.

[15] E. Nardelli, G. Proietti, and P. Widmayer, A faster computation of the most vital edge of a shortest path, *Inform. Process. Lett.*, **79**(2) (2001), 81–85.

[16] E. Nardelli, G. Proietti, and P. Widmayer, Finding the most vital node of a shortest path, *Theoret. Comput. Sci.*, **296**(1) (2003), 167–177. A preliminary version appeared in *Proc. of the* 7*th Annual International Computing and Combinatorics Conference* (*COCOON* '01), Vol. 2108 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 278–287.

[17] N. Nisan and A. Ronen, Algorithmic mechanism design, *Games Econom. Behaviour*, **35** (2001), 166–196. A preliminary version appeared in *Proc. of the* 31*st Annual ACM Symposium on Theory of Computing* (*STOC* 1999), pp. 129–140.

[18] D.D. Sleator and R.E. Tarjan, Self-adjusting heaps, *SIAM J. Comput.*, **15**(1) (1986), 52–69.

[19] R.E. Tarjan, Efficiency of a good but not linear set union algorithm, *J. Assoc. Comput. Mach.*, **22**(2) (1975), 215–225.

[20] R.E. Tarjan, Applications of path compression on balanced trees, *J. Assoc. Comput. Mach.*, **26**(4) (1979), 690–715.

[21] W. Vickrey, Counterspeculation, auctions and competitive sealed tenders, *J. Finance*, **16** (1961), 8–37.