Algorithmica (2003) 35: 56–74
DOI: 10.1007/s00453-002-0988-z

Algorithmica

Swapping a Failing Edge of a Single Source Shortest Paths Tree Is Good and Fast[1]

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

Abstract. Let $G = (V, E)$ be a 2-edge connected, undirected and nonnegatively weighted graph, and let $S(r)$ be a single source shortest paths tree (SPT) of G rooted at $r \in V$. Whenever an edge e in $S(r)$ fails, we are interested in reconnecting the nodes now disconnected from the root by means of a single edge e' crossing the cut created by the removal of e. Such an edge e' is named a *swap edge* for e. Let $S_{e/e'}(r)$ be the *swap tree* (no longer an SPT, in general) obtained by swapping e with e', and let $\mathcal{S}_e$ be the set of all possible swap trees with respect to e. Let F be a function defined over $\mathcal{S}_e$ that expresses some feature of a swap tree, such as the average length of a path from the root r to all the nodes below edge e, or the maximum length, or one of many others. A *best swap edge* for e with respect to F is a swap edge f such that $F(S_{e/f}(r))$ is minimum.

In this paper we present efficient algorithms for the problem of finding a best swap edge, for each edge e of $S(r)$, with respect to several objectives. Our work is motivated by a scenario in which individual connections in a communication network suffer transient failures. As a consequence of an edge failure, the shortest paths to all the nodes below the failed edge might completely change, and it might be desirable to avoid an expensive switch to a new SPT, because the failure is only temporary. As an aside, what we get is not even far from a new SPT: our analysis shows that the trees obtained from the swapping have features very similar to those of the corresponding SPTs rebuilt from scratch.

Key Words. Network survivability, Single source shortest paths tree, Swap algorithms.

1. Introduction. Survivability of a communication network denotes the ability of the network to remain operational even if individual network components (such as a link or even a node) fail. In the past few years, several survivability problems have been studied intensely [4]. 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 more vulnerable to failures, in the

[1] This work has been partially supported by the Research Training Network Contract No. HPRN-CT-1999-00104 funded by the European Union, by the CNR-Agenzia 2000 Program, under Grants Nos. CNRC00CAB8 and CNRG003EF8, and by the Research Project REAL-WINE, partially funded by the Italian Ministry of Education, University and Research. A preliminary version of this paper appeared in the *Proceedings of the 5th Annual International Computing and Combinatorics Conference* (*COCOON* '99), Vol. 1627 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1999, pp. 144–153.

[2] 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. {nardelli, proietti}@di.univaq.it.

[3] Institut für Theoretische Informatik, ETH Zentrum, CLW C 2, Clausiusstrasse 49, 8092 Zürich, Switzerland. widmayer@inf.ethz.ch.

Received July 31, 2001; revised June 6, 2002. Communicated by T. Nishizeki.
Online publication October 29, 2002.

sense that each link failure might disconnect the network. Therefore, it is important for sparse networks also to take survivability into account from the very beginning.

Assuming that a link that is down comes back up quickly, the likelihood of having many failures at the same time is quite small. Therefore, we study the problem of dealing in isolation with the failure of each single link in the network: since we can expect that sooner or later each link will fail, it is preferable to perform a single global computation on the network, to know in advance how to manage a failure. In a practical setting, handling a failing link entails all kinds of accompanying measures, such as a change in the routing pattern, in addition to finding an alternative network configuration which does not make use of the failed link. To keep these secondary measures cheap, it is desirable to find a most simple alternative network configuration; in the simplest case, a single new link is used to make up for the failing link. In this paper this is exactly the problem we study: given the restriction that a failing edge of a graph can be compensated only by a single extra, new edge, that we call a *swap edge*, how should the swap edge be chosen? For example, if the network is a minimum spanning tree of a given weighted graph, then the optimum might be to choose a swap edge of minimum weight. For minimum spanning trees, this question has been studied before [1], [8], [13]. In this paper we study the corresponding question for shortest paths trees, by extending our preliminary results presented in [10].

1.1. *Basic Definitions.* We first recall some of the basic graph terminology that we use in what follows; for details, see [5]. Let $G = (V, E)$ be an undirected graph, where V is the set of nodes 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 nodes and the number of edges, respectively. A graph $H = (V(H), E(H))$ is a *subgraph* of G if $V(H) \subseteq V$ and $E(H) \subseteq E$. If $V(H) \equiv V$, then H is a *spanning* subgraph of G.

A *simple path* (or a *path* for short) in G is a subgraph P with $V(P) = \{v_1, \ldots, v_k \mid v_i \neq v_j \text{ for } i \neq j\}$ and $E(P) = \{(v_i, v_{i+1}) \mid 1 \leq i < k\}$, also denoted as $P = \langle v_1, \ldots, v_k \rangle$. Path P is said to go from v_1 to v_k and to have *length* equal to the sum of the lengths of its edges. If $v_1 \equiv v_k$, then P is a *cycle*. A graph G is *connected* if, for any two nodes $u, v \in V$, there exists a path in G going from u to v. A graph G is 2-*edge connected* if at least two edges in G must be removed to disconnect G.

A *tree* is a connected graph which does not contain a cycle. Let T be a spanning tree of a 2-edge connected graph G. A *swap edge* for an edge $e = (u, v) \in E(T)$ is an edge $e' = (u', v') \in E \backslash \{e\}$ reconnecting the two subtrees S_u and S_v created by the removal of e, where in here and in the rest of the paper it is assumed that $u, u' \in V(S_u)$ and $v, v' \in V(S_v)$. Let in the following $T_{e/e'}$ denote the *swap tree* obtained by swapping e with e', and let $\mathcal{S}_e$ be the set of all possible swap trees with respect to e. Depending on the goal that we pursue in swapping, some swap edge may be preferable to some other one. We capture this objective by defining a nonnegative real function F over $\mathcal{S}_e$. Intuitively, the function F represents the objective of the optimization in the swap. A *best swap edge* for an edge e is a swap edge f such that $F(T_{e/f})$ is minimum. A *swap algorithm* is an algorithm that finds a best swap edge for every edge $e \in E(T)$. All the swap algorithms that are presented in this paper are assumed to be implemented on a comparison-based model of computation.

1.2. *Related Work.* Coping with a failure of a link in a network T that is a spanning tree of a given 2-edge connected graph G means, on the theoretical side, to define an

interesting family of problems on graphs. For example, if T is a *minimum spanning tree* (MST) of G, that is, a spanning tree whose sum of edge lengths is *minimum* among all the spanning trees of G, then the natural function to be defined is $F(T_{e/e'}) = \sum_{g \in E(T_{e/e'})} |g|$. Therefore, a best swap edge for a failing edge e is an edge f such that $T_{e/f}$ is a spanning tree of minimum length among all possible swap edges. It is easy to see that here f is simply a swap edge of minimum length. In this particular case, $T_{e/f}$ coincides with the MST T'_e of $G - e = (V, E\backslash\{e\})$. Therefore, an algorithm finding a best swap for each edge in T can be viewed as a kind of dynamic algorithm dealing with the particular instance of the deletion and the subsequent reinsertion of each edge in T. The fastest solution for finding a best swap edge for each edge in T runs in $O(m)$ time [1].

Another interesting problem arises when T is a *minimum diameter spanning tree* (MDST) of G, that is, a spanning tree having the maximum length of a path on the tree between any two nodes *minimum* among all the spanning trees of G. In this case the natural function to be defined is the diameter of $T_{e/e'}$, and a best swap edge is a swap edge which makes the diameter of the new spanning tree as low as possible (for practical motivations, see [7]). The problem of finding a best swap edge for each edge in T can be solved in $O(n\sqrt{m})$ time and $O(m)$ space [9]. Notice that in this case $T_{e/f}$ does not coincide with the MDST T'_e of $G - e$. However, it can be shown that the diameter of $T_{e/f}$ is at most $5/2$ times the diameter of T'_e. Therefore, $T_{e/f}$ is *functionally* close to T'_e.

1.3. *Single Source Shortest Paths Trees.* Among all paths between two nodes $v, v' \in V$, a path is called *shortest* if the length of the path is smallest. Let the *distance* $d(v, v')$ between two nodes v, v' in G be the length of a shortest path between v and v'. For a distinguished node $r \in V$, called the *source*, and all the nodes $v \in V\backslash\{r\}$, a *shortest paths tree* (SPT) $S(r)$ rooted at r is a spanning tree of G formed by the union of shortest paths, with one shortest path from r to v for each $v \in V\backslash\{r\}$. To simplify notations, we denote by E_S the set of edges of $S(r)$. Figure 1 illustrates an SPT $S(r)$ of a graph G, along with the set of swap edges for a given edge $e = (u, v) \in E_S$.

Many network architectures are based on an SPT. In fact, this is especially interesting as a model for a (centralized) network where a (privileged) node broadcasts messages to all the other nodes. In this case no single, unique and natural notion of best swap edge for a failing edge $e = (u, v)$, with u closer to r than v, is as obvious as for the

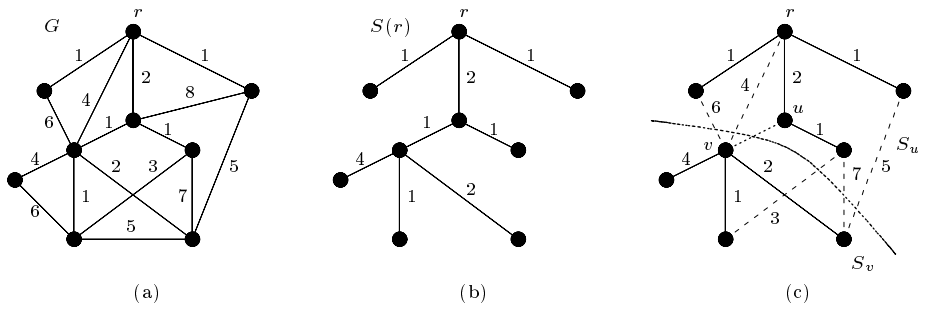

Fig. 1. (a) A weighted graph $G = (V, E)$; (b) an SPT $S(r)$ rooted at r; (c) edge $e = (u, v)$ is removed from $S(r)$: dashed edges are swap edges.

MST and the MDST. For example, here one could be interested in choosing a swap edge e' that keeps as low as possible either the maximum or the average distance in $S_{e/e'}(r)$ between r and any node descending from v in $S(r)$, among other interesting alternatives from a network management point of view. Therefore, a comprehensive study of the problem requires a rigorous definition of a number of objective functions to be minimized. Hence, the algorithm for finding a best swap edge for any edge in $S(r)$ will depend on the function F, and so will its complexity. In this paper we propose efficient swap algorithms for several functions expressing desirable features of an SPT, as specified in detail in Section 2. These algorithms are much faster than recomputing for each edge $e \in E_S$ from scratch an SPT $S'_e(r)$ of $G - e$. It is fair to compare with a recomputation from scratch, since no dynamic solution for this problem is known to be asymptotically better [3].

Furthermore, we compare $S_{e/f}(r)$ and $S'_e(r)$, on the basis of those features captured by the functions that have been considered for the swapping. More precisely, if a swap tree is obtained starting from a function expressing a given feature, then we measure such a feature in $S_{e/e'}(r)$ and in $S'_e(r)$. Moreover, to make the comparison more expressive, we go one step further: we also study the effect that a swap algorithm has on the other features that it does not aim at. Somewhat surprisingly, for some swap algorithms, the ratio of a given feature, as measured in $S_{e/f}(r)$ and in $S'_e(r)$, stays within a small constant factor (in the worst case) for all the considered features. That is, the swap is a good solution to a problem at which it does not even aim! Hence, swapping can be viewed as fast and good at the same time.

The paper is organized as follows. In Section 2 we define more precisely and formally the problems we are dealing with. Section 3 proposes the algorithms for solving the problems. In Section 4 we present a comparison between the solutions computed by the various swap algorithms and the respective exact solutions. Finally, in Section 5, we discuss the obtained results and list some possible extensions and open problems.

2. The Swap Problems.

A *swap problem* with respect to a given function F asks for finding a best swap edge (with respect to F) for every edge $e \in E_S$. In this section we formally define the set of swap problems that are considered throughout the paper.

2.1. *One-to-One Swap Problems*.

A first class of swap problems, that we call *one-to-one swap problems*, arises when F simply focuses on two fixed nodes, one node s in S_u and one node t in S_v. Since only two nodes are involved, it is quite natural to let F be the distance between the two nodes. More formally, let in the following $d_{e/e'}(v, v')$ denote the distance in $S_{e/e'}(r)$ between any two nodes v and v', and let $\mathcal{C}_e$ denote the set of all possible swap edges for e. Then we set

$$F_{\{s,t\}}(S_{e/e'}(r)) = d_{e/e'}(s, t),$$
$$\text{with} \quad e = (u, v) \in E_S, \quad e' \in \mathcal{C}_e, \quad s \in V(S_u) \quad \text{and} \quad t \in V(S_v).$$

Depending on the identity of s and t, a set of one-to-one problems is defined. Quite naturally, the most meaningful choices for s and t are those expressed by the following couple of one-to-one swap problems:

$\{r, v\}$-PROBLEM. In this problem, a best swap edge minimizes the distance from the root r of $S(r)$ to the root v of S_v. Solving this problem efficiently has applications in network broadcasting. To make a concrete example, one could imagine a situation in which a message broadcasted from the root is processed by each node in the network, and a node can process the message only after its parent in $S(r)$ has done that. Therefore, after the edge e has failed, if the node u sent back to r the message partially processed up to node u itself, then the root must route such a message to v as fast as possible.

$\{u, v\}$-PROBLEM. In this problem, a best swap edge minimizes the distance from u to the root v of S_v. This is motivated since the failure of edge e is detected by node u, and u wants to send its message to v as fast as possible.

It is worth noting that these two problems do not exhaust the possible choices for s and t, and alternative choices can be motivated on the basis of different network functionalities.

2.2. *One-to-Many Swap Problems*. A second class of swap problems, that we call *one-to-many swap problems*, arises when the function F focuses on a fixed node $s \in V(S_u)$ and on a subset of nodes $\mathcal{T} \subseteq V(S_v)$. Once again, depending on F and on the identity of s and $\mathcal{T}$, several different one-to-many swap problems can be defined. Concerning s, as for one-to-one problems, it is natural to restrict our attention to r and u. On the other hand, with regard to $\mathcal{T}$, given that SPTs are particularly used for broadcasting (and therefore all the nodes in S_v deserve the same attention), it is natural to focus on S_v as a whole. Concerning the function F, given that many nodes are now involved, there is not just a single, natural way to define it. However, since a reasonable function must be related with the structural properties of an SPT, we can claim that, for $e = (u, v) \in E_S$, $e' \in \mathcal{C}_e$ and $s \in V(S_u)$, the following functions are well representative:

1. $F_{\{s,\Sigma\}}(S_{e/e'}(r)) = \sum_{t \in V(S_v)} d_{e/e'}(s, t)$;
2. $F_{\{s,\Delta\}}(S_{e/e'}(r)) = \max\{d_{e/e'}(s, t) - d(s, t)\colon\ t \in V(S_v)\}$;
3. $F_{\{s,\min\}}(S_{e/e'}(r)) = \min\{d_{e/e'}(s, t)\colon\ t \in V(S_v)\}$;
4. $F_{\{s,\max\}}(S_{e/e'}(r)) = \max\{d_{e/e'}(s, t)\colon\ t \in V(S_v)\}$.

Therefore, when focusing on the root r, the following one-to-many swap problems arise:

$\{r, \Sigma\}$-PROBLEM. In this problem, a best swap edge minimizes the function $F_{\{r,\Sigma\}}$, that is, it is a swap edge minimizing the total distance from r to all the nodes in S_v. Choosing such a swap edge will therefore minimize the average delay in delivering a message from the root to any node in S_v.

$\{r, \Delta\}$-PROBLEM. In this problem, a best swap edge minimizes the function $F_{\{r,\Delta\}}$, that is, it is a swap edge minimizing the maximum increment of the distance from r to any node in S_v. Choosing such a swap edge will therefore minimize the maximum "moving away" of a node with respect to the root.

$\{r, \min\}$-PROBLEM. In this problem, a best swap edge minimizes the function $F_{\{r,\min\}}$, that is, it is a swap edge $f = (x, y)$ such that the distance from r to y is minimum. Choosing such a swap edge will therefore minimize the delay in delivering a message from the root to a node in S_v.

$\{r, \max\}$-PROBLEM. In this problem, a best swap edge minimizes the function $F_{\{r,\max\}}$, that is, it is a swap edge minimizing the longest path from r to any node in S_v. Choosing such a swap edge will therefore minimize the maximum delay in delivering a message from the root to any node in S_v.

Similarly, when focusing on node u, the $\{u, \Sigma\}$-Problem, the $\{u, \Delta\}$-Problem, the $\{u, \min\}$-Problem and the $\{u, \max\}$-Problem are defined.

3. The Swap Algorithms. In this section we present efficient swap algorithms for all the problems listed in the previous section. These algorithms make use of a set of techniques, which can be tailored on the problem that needs to be solved. To make the presentation more concise, we present in detail only the solutions for all the problems in which the root r is considered, while their variations for the case in which u takes the place of r are given in the Appendix.

3.1. *One-to-One Swap Algorithms*

3.1.1. *Solving the $\{r, v\}$-Problem.* Remember that the $\{r, v\}$-Problem asks for a swap edge minimizing the distance from r to v, when the edge $e = (u, v)$ fails. A brute-force approach for solving this problem would require the inspection, for each failing edge e of $S(r)$, of the $O(m)$ edges in $\mathcal{C}_e$: for each swap edge e', we can compute in $O(1)$ time $d_{e/e'}(r, v)$, and thus we spend $O(m)$ time to select a best swap edge, i.e., a total $O(n \cdot m)$ time to solve the problem. We now show how the above time bound can be improved substantially. Let $\alpha(m, n)$ denote the functional inverse of the Ackermann function [11]. The following can be proved:

THEOREM 3.1. *There exists a swap algorithm solving the $\{r, v\}$-Problem in $O(m \cdot \alpha(m, n))$ time and $O(m)$ space.*

PROOF. A swap algorithm needs to process all the edges in $\mathcal{C}_e$, for any $e \in E_S$. To do that, we make use of a *transmuter* [13]. A transmuter $D_G(T)$ is a *directed acyclic graph* that represents the set of fundamental cycles of a graph G with respect to a spanning tree T. Basically, $D_G(T)$ contains one source node $s(e)$ representing each tree edge e, one sink node $t(e')$ representing each nontree edge e', and an arbitrary number of additional nodes. The fundamental property of a transmuter is that there is a path from a given source $s(e)$ to a given sink $t(e')$ if and only if e and e' form a cycle in T. It is clear that in $S(r)$, all and only the edges belonging to $\mathcal{C}_e$ form a cycle with e. Therefore, we can build a transmuter having as source nodes all the edges belonging to $S(r)$ and as sink nodes all the nontree edges. This can be done in $O(m \cdot \alpha(m, n))$ time and $O(m)$ space [14]. To associate e with its best swap edge, it remains to establish the value that has

to be given to a sink node. Since we have to minimize the value of $d_{e/e'}(r, v), \forall e' \in \mathcal{C}_e$, it follows that a best swap edge will be an edge $f = (x, y)$ such that

$$(1) \quad d(r, x) + |f| + d(v, y) = \min\{d(r, u') + |e'| + d(v, v') : e' = (u', v') \in \mathcal{C}_e\}.$$

If $e' = (u', v') \in \mathcal{C}_e$ forms a cycle with $e_1 = (u_0 \equiv v', u_1), e_2 = (u_1, u_2), \ldots, e_k = (u_{k-1}, u_k \equiv u'), e_i \in E_S, i = 1 \ldots k$, then the value of (1) depends on which e_i is considered. Hence, the question is: which value $c(e')$ in the transmuter should be associated with e' such that $c(e')$ is *independent* of e_i? To overcome this problem, we associate in $O(1)$ time with e' the length of the (not simple) cycle in $S(r)$ starting from r and passing through e', that is,

$$c(e') = d(r, u') + |e'| + d(r, v').$$

In fact, $d_{e/e'}(r, v) = c(e') - d(r, v)$ for any edge $e' \in \mathcal{C}_e$ and, therefore, with a shortest cycle is associated a best swap edge, and vice versa. Hence, we can solve the $\{r, v\}$-Problem by processing the nodes of the transmuter in reverse topological order, labeling each node with the minimum of the labels of its immediate successors. When the process is complete, each source node $s(e)$ is labeled with the minimum cost $c(f)$ of a best swap edge $f \in \mathcal{C}_e$. This completes the proof. □

3.1.2. *Solving the* $\{u, v\}$*-Problem.* Remember that the $\{u, v\}$-Problem asks for a swap edge minimizing the distance from u to v, when the edge (u, v) fails. A brute-force approach for solving this problem would require the inspection, for each failing edge e of $S(r)$, of the $O(m)$ edges in $\mathcal{C}_e$: for each swap edge $e' = (u', v')$, let $z_{e'}$ be the *nearest common ancestor* in $S(r)$ of u' and v'. Recall that the nearest common ancestors of all nontree edges can be computed in $O(m \cdot \alpha(m, n))$ time and $O(n)$ space [6]. Since

$$d_{e/e'}(u, v) = d(r, u') + d(r, v') + |e'| - |e| - 2d(r, z_{e'}),$$

we can compute in $O(1)$ time $d_{e/e'}(u, v)$, and thus we spend $O(m)$ time to select a best swap edge. Hence, it follows that we spend a total $O(n \cdot m + m \cdot \alpha(m, n)) = O(n \cdot m)$ time to solve the problem. However, similarly to the $\{r, v\}$-Problem, the following improvement can be proved:

THEOREM 3.2. *There exists a swap algorithm solving the* $\{u, v\}$*-Problem in* $O(m \cdot \alpha(m, n))$ *time and* $O(m)$ *space.*

PROOF. See the Appendix. □

3.2. *One-to-Many Swap Algorithms*

3.2.1. *Solving the* $\{r, \Sigma\}$*-Problem.* Remember that the $\{r, \Sigma\}$-Problem asks for a swap edge minimizing the sum of the lengths of all the paths in the swap tree from r to each node in S_v, when the edge (u, v) fails. A straightforward solution for this problem is the following: for each failing edge e of $S(r)$, consider the $O(m)$ edges in $\mathcal{C}_e$; for each swap edge e', compute in $O(n)$ time the sum of the lengths of all the paths starting from r,

passing through e' and leading to a node in S_v, and finally select an edge minimizing such a sum. Hence, the total time spent is $O(n^2 \cdot m)$.

We now show how the above time bound can be improved to $O(n^2)$. Let $v \in V$ and let t be any node in S_v. First, we define the auxiliary values associated by the algorithm with v:

- $|S_v|$: number of nodes in S_v;
- $down(v)$: total length of all the paths in S_v starting from v; if v is a leaf, then set $down(v) = 0$;
- $up(t, v)$: total length of all the paths in S_v starting from t and leading to nodes which are not descendants of t in $S(r)$;
- $min\_path(t, v)$: after the removal of the edge $e = (u, v)$ in $S(r)$, this is the length of a shortest path from r to t making use of edges in S_u plus a swap edge leading to t. More formally

$$min\_path(t, v) = \min\{d(r, u') + |e'|:\ e' = (u', t) \in \mathcal{C}_e\}. \tag{2}$$

  If no such edge e' exists, then set $min\_path(t, v) = +\infty$;
- $all\_paths(t, v)$: after the removal of the edge $e = (u, v)$ in $S(r)$, this is the total length of all the paths starting from r, making use of edges of S_u, passing through the edge selected as specified at the previous item and leading to nodes in S_v. It is not hard to see that

$$all\_paths(t, v) = min\_path(t, v) \cdot |S_v| + down(t) + up(t, v). \tag{3}$$

Figure 2 shows an example in which the above values are computed for two fixed nodes t, v.

A high-level description of our algorithm for solving this problem is the following. We consider all the edges of $S(r)$ in any arbitrary postorder. We now fix such an edge $e = (u, v)$. For each node t in S_v, we compute $all\_paths(t, v)$, as specified above; then we select the minimum over these values for all the nodes t in S_v and we return the corresponding best swap edge. The crucial point is the efficient computation of $all\_paths(t, v)$, which in its turn requires the computation of $min\_path(t, v)$ and $up(t, v)$. The former is computed in advance by computing, for each $v \in V$, a selected swap edge (i.e., a swap edge minimizing (2)) for any of its ancestors in $S(r)$, while the latter is computed by keeping track, during the postorder traversal, of the total length of all the paths from t that stay within S_v. More specifically, the algorithm is the following:

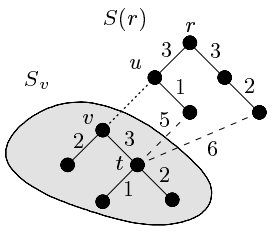

Fig. 2. A weighted graph and an SPT $S(r)$ (solid edges); nontree edges are dashed, while the removed edge is dotted. Then $|S_v| = 5$, $down(t) = 1 + 2$, $up(t, v) = 3 + 5$, $min\_path(t, v) = 9$ (using the swap edge of length 5), $all\_paths(t, v) = 9 + 10 + 11 + 12 + 15 = 56$, which coincides with definition (3).

Algorithm SIGMA(G, $S(r)$)

Input: A weighted, 2-edge connected graph $G = (V, E)$ and an SPT $S(r) = (V, E_S)$;
Output: $\forall e = (u, v) \in E_S$, a swap edge f such that $\sum_{t \in V(S_v)} d_{e/f}(r, t) = \min\{\sum_{t \in V(S_v)} d_{e/e'}(r, t)\colon\ e' \in \mathcal{C}_e\}$.

```
Step 1.  For each node v ∈ V as considered by any postorder visit
Step 2.      Compute down(v);
Step 3.      For each ancestor w ≠ r of v (including v) compute
             min_path(v, w);
Step 4.  For each edge e=(u, v) ∈ E_S as considered by any postorder
         visit
Step 5.      For each children v_i of v
Step 6.          up(v_i, v) = down(v) − down(v_i) + (|S_v| − |S_{v_i}| − 1)
                 ·|(v_i, v)|;
Step 7.          For each node t ∈ V(S_{v_i}), set up(t, v) = up(t, v_i) +
                 (|S_v| − |S_{v_i}|) · d(v_i, t) + up(v_i, v);
Step 8.      For each node t ∈ V(S_v), set all_paths(t, v) = min_path(t, v)
             ·|S_v| + down(t) + up(t, v);
Step 9.      Compute t_min such that all_paths(t_min) = min{all_paths(t, v):
             t ∈ V(S_v)};
Step 10.     Return the edge associated with min_path(t_min, v).
```

THEOREM 3.3. *The swap algorithm* SIGMA(G, $S(r)$) *solves the* $\{r, \Sigma\}$*-Problem in* $O(n^2)$ *time and space.*

PROOF. The correctness of the algorithm is a consequence of the fact that it considers exhaustively at each step all the possible best swap edges. To establish the running time and the space requirements of the algorithm, we analyze it step by step.

Concerning Step 2, we can compute $down(v)$ in $O(n)$ time and space by simply performing a postorder visit of $S(r)$.

Step 3 can be accomplished in $O(n^2)$ time and space for all the nodes in the following way: let $\langle r \equiv w_0, w_1, \ldots, w_k \equiv v\rangle$ be the path in $S(r)$ joining r and v. We start by bucketing the nontree edges adjacent to v with respect to their nearest common ancestors. This can be done in $O(\alpha(m, n))$ amortized time for each edge [6], that is, it will cost $O(m \cdot \alpha(m, n))$ time for all the nodes. Let $b(v, w_i)$ be the bucket containing the edges associated with $w_i, i = 0, \ldots, k-1$. We initially search for the edge in $b(v, w_0)$ minimizing the path from r to v. This can be done in time and space proportional to the number of elements in $b(v, w_0)$. This value defines $min\_path(v, w_1)$. Afterwards, we repeat the step for $b(v, w_1)$: if the found value is less than $min\_path(v, w_1)$, then this becomes $min\_path(v, w_2)$, otherwise we set $min\_path(v, w_2) = min\_path(v, w_1)$. The process goes on iteratively, up to $b(v, w_{k-1})$. This way, we spend $O(n)$ time and space for each node to process the buckets. Thus, since $O(\alpha(n^2, n)) = O(1)$, then Step 3 can be done in $O(n^2 + m \cdot \alpha(m, n)) = O(n^2)$ time and space for all the nodes.

Step 6 can be accomplished in $O(1)$ time for each node and in $O(n)$ total time for all the nodes. Steps 7–9 can be executed in $O(n)$ time, and therefore they require a total

$O(n^2)$ time for all the nodes. Finally, Step 10 costs $O(1)$ time per node. Therefore, the overall running time and the space requirements are $O(n^2)$. □

3.2.2. *Solving the* $\{r, \Delta\}$*-Problem.* Remember that the $\{r, \Delta\}$-Problem asks for a swap edge minimizing the maximum increase of the distance from r to any node in S_v, when the edge $e = (u, v)$ fails. Similarly to the $\{r, \Sigma\}$-Problem, a brute-force solution for solving this problem would require $O(n^2 \cdot m)$ time (in fact, rather than computing the sum of the path lengths, we now check the increment of the length of each path). However, the following result improving the above time bound can be proved:

THEOREM 3.4. *There exists a swap algorithm solving the* $\{r, \Delta\}$*-Problem in* $O(m \cdot \alpha(m, n))$ *time and* $O(m)$ *space.*

PROOF. A best swap edge by definition is an edge f such that

$$\max\{d_{e/f}(r, t) - d(r, t)\colon\ t \in V(S_v)\} = \min\{\max\{d_{e/e'}(r, t) - d(r, t)\colon\ t \in V(S_v)\}\colon\ e' \in \mathcal{C}_e\}. \tag{4}$$

For any swap edge e' and for any node $t \in V(S_v)$ we have

$$\begin{aligned} d_{e/e'}(r, t) - d(r, t) &\leq d_{e/e'}(r, v) + d_{e/e'}(v, t) - d(r, t) \\ &= d_{e/e'}(r, v) + d(v, t) - d(r, t) = d_{e/e'}(r, v) - d(r, v). \end{aligned}$$

Then (4) becomes $d_{e/f}(r, v) - d(r, v) = \min\{d_{e/e'}(r, v) - d(r, v)\colon\ e' \in \mathcal{C}_e\}$, and therefore

$$d_{e/f}(r, v) = \min\{d_{e/e'}(r, v)\colon\ e' \in \mathcal{C}_e\}.$$

Hence, the $\{r, \Delta\}$-Problem reduces to the $\{r, v\}$-Problem, and therefore it can be solved using the approach described in Theorem 3.1. This completes the proof. □

3.2.3. *Solving the* $\{r, \min\}$*-Problem.* Remember that the $\{r, \min\}$-Problem asks for a swap edge $e' \in \mathcal{C}_e$ minimizing the distance from r to a node in S_v, when the edge $e = (u, v)$ fails. Trivially, the node in S_v closest to r as a consequence of the swap with $e' = (u', v')$ is v', and then we have to minimize $d_{e/e'}(r, v')$ over all the swap edges. Hence, similarly to the $\{r, v\}$-Problem, a brute-force approach for solving this problem would require $O(n \cdot m)$ time, since for each swap edge e', we can compute in $O(1)$ time $d_{e/e'}(r, v')$, thus spending $O(m)$ time for each failing edge to select a best swap edge. We now show that the above time bound can be improved:

THEOREM 3.5. *There exists a swap algorithm solving the* $\{r, \min\}$*-Problem in* $O(m \cdot \alpha(m, n))$ *time and* $O(m)$ *space.*

PROOF. To associate e with its best swap edge f, we have to minimize the value of $d_{e/e'}(r, v'), \forall e' \in \mathcal{C}_e$. It follows that a best swap edge will be an edge $f = (x, y)$ such that

$$d(r, x) + |f| = \min\{d(r, u') + |e'|\colon\ e' = (u', v') \in \mathcal{C}_e\}. \tag{5}$$

Notice that if $e' = (u', v') \in \mathcal{C}_e$ forms a cycle with $e_1 = (u_0 \equiv v', u_1), e_2 =$

$(u_1, u_2), \ldots, e_k = (u_{k-1}, u_k \equiv u')$, $e_i \in E_S$, $i = 1 \cdots k$, then the value of (5) depends on which e_i is considered, and then eventually it depends on (u, v). More precisely, let $z_{e'} \equiv u_i, 0 \leq i \leq k$, be the nearest common ancestor of u' and v' in $S(r)$; then we have that edge e' remains associated either with the value $d(r, u') + |e'|$, whenever an edge e_j, $j = 1, \ldots, i$, fails, or with the value $d(r, v') + |e'|$, whenever an edge e_j, $j = i + 1, \ldots, k$, fails. Therefore, to solve this problem, we simply substitute each nontree edge e' for which the corresponding nearest common ancestor does not coincide with one of its endnodes, with a couple of edges $r_1(e') = (z_{e'}, u')$ and $r_2(e') = (z_{e'}, v')$, of weight $|e'|$. In this way, the original graph G is transformed to a graph $G' = (V, E')$ with the same nodes and with at most $2m$ edges. Moreover, it is easy to see that an edge f is a best swap edge for a given edge e in G if and only if either $r_1(f)$ or $r_2(f)$ is a best swap edge for e in G'. Since $z_{e'}$ can be found in $O(\alpha(m, n))$ amortized time for each edge $e' \in E \backslash E_S$ [6], we have that G' can be computed in $O(m)$ time. Then we build a transmuter $D_{G'}(S(r))$ in which a sink node associated with a nontree edge $e' = (u', v')$, such that $z_{e'} \equiv u'$, is labeled with $c(e') = d(r, v') + |e'|$, not depending on (u, v). Finally, we can solve the $\{r, \min\}$-Problem by processing the nodes of the transmuter in reverse topological order. This completes the proof. □

3.2.4. *Solving the* $\{r, \max\}$*-Problem.* Remember that the $\{r, \max\}$-Problem asks for a swap edge minimizing the length of a longest path starting from r and ending in S_v. Similarly to the $\{r, \Sigma\}$-Problem, a brute-force solution for solving this problem would require $O(n^2 \cdot m)$ time (in fact, rather than computing the sum of the path lengths, we now check the length of each path). However, the above time bound can be lowered, as the following theorem shows:

THEOREM 3.6. *There exists a swap algorithm solving the* $\{r, \max\}$*-Problem in* $O(n\sqrt{m})$ *time and* $O(m)$ *space.*

PROOF. This problem can be solved by slightly modifying the approach used in [9], where the problem of computing all the best swap edges for a minimum diameter spanning tree has been solved. In fact, as a subroutine of the main algorithm, the length of a longest path starting from $t \in V(S_v)$ and staying within S_v (which is exactly what we need, once that we add $d_{e/e'}(r, t)$) is there computed, and this costs $O(n\sqrt{m})$ time and $O(m)$ space. □

3.2.5. *Solving the One-to-Many Swap Problems When u Takes the Place of r.* The one-to-many swap problems in which u takes the place of r can be similarly treated. More precisely, the following result holds:

THEOREM 3.7. *There exist swap algorithms solving*:

1. *the* $\{u, \Sigma\}$*-Problem in* $O(n^2)$ *time and space*;
2. *the* $\{u, \Delta\}$*-Problem in* $O(m \cdot \alpha(m, n))$ *time and* $O(m)$ *space*;
3. *the* $\{u, \min\}$*-Problem in* $O(m \cdot \alpha(m, n))$ *time and* $O(m)$ *space*;
4. *the* $\{u, \max\}$*-Problem in* $O(n\sqrt{m})$ *time and* $O(m)$ *space.*

PROOF. See the Appendix. □

4. Swapping versus Recomputing from Scratch. Since swapping a single edge for a failed one is fast and involves very few changes in the underlying network (e.g., as to routing information) it is interesting to see how the tree obtained from swapping compares (according to some of the functions we have introduced) with a true SPT $S'_e(r)$ of $G - e$. It is worth noting that recomputing from scratch $S'_e(r)$ for each failing edge $e \in E_S$ is much more costly than applying any swap algorithm, since no dynamic solution for this problem is known to be asymptotically better than recomputing n times an SPT [3]. Since the fastest algorithm for computing an SPT in a comparison-based model requires $O(m + n \log n)$ time [2], it follows that recomputing from scratch $S'_e(r)$ for each failing edge $e \in E_S$ results in an $O(mn + n^2 \log n)$ time algorithm.

Although it might appear unfair to compare $S'_e(r)$ and $S_{e/f}(r)$, since the latter might seem at a first glance quite different from the former, we will show that this is not the case, at least for some of the swap algorithms. However, what does it mean *to compare two trees*? In fact, while $S'_e(r)$ is well defined, this is not the case for $S_{e/f}(r)$, since this depends on the function whose minimization it aimed at. In particular, for the functions involving u, it appears more reasonable to perform a comparison between $S_{e/f}(r)$ and $S'_e(u)$, an SPT of $G - e$ rooted at u. Therefore, we here restrict our attention to all the problems in which the root r is concerned. Moreover, since the $\{r, \Delta\}$-Problem and the $\{r, v\}$-Problem are equivalent, we analyze the features captured by the functions associated with the $\{r, \Sigma\}$-Problem, the $\{r, \Delta\}$-Problem, the $\{r, \min\}$-Problem and the $\{r, \max\}$-Problem, respectively.

While it is natural to study each of the four quality criteria (functions) for the algorithms that optimize the corresponding swap, we go one step further: we also study the effect that a swap algorithm has on the other criteria (that it does not aim at). This will give us a general idea of the effectiveness of each swap algorithm. Moreover, in a general context in which a balancing of the above criteria is needed, one is allowed to perform a satisfying choice. Let $d'(v, v')$ denote the distance in $S'_e(r)$ between any two nodes v and v'. For each of the above swap algorithm, we therefore consider the following ratios in the two trees:

$$\rho_\Sigma = \frac{\sum_{t \in V(S_v)} d_{e/f}(r, t)}{\sum_{t \in V(S_v)} d'(r, t)}; \qquad \rho_\Delta = \frac{d_{e/f}(r, v)}{d'(r, v)};$$

$$\rho_{\min} = \frac{\min\{d_{e/f}(r, t)\colon\ t \in V(S_v)\}}{\min\{d'(r, t)\colon\ t \in V(S_v)\}}; \qquad \rho_{\max} = \frac{\max\{d_{e/f}(r, t)\colon\ t \in V(S_v)\}}{\max\{d'(r, t)\colon\ t \in V(S_v)\}}.$$

In what follows, $h(S_v)$ denotes the *height* of S_v, that is, the length of a longest path between v and any node in S_v, while ℓ denotes the height of $S(r)$ *restricted* to S_v, that is, the length of a longest path in $S(r)$ between r and any node in S_v. Similarly, ℓ' and $\ell_{e/f}$ denote the heights of $S'_e(r)$ and $S_{e/f}(r)$ restricted to S_v. Note that $h(S_v) \leq \ell$, $h(S_v) \leq \ell'$ and $h(S_v) \leq \ell_{e/f}$.

4.1. *Ratios for the Swap Algorithm Solving the* $\{r, \Sigma\}$*-Problem.* We can prove the following result:

THEOREM 4.1. *For the swap algorithm solving the* $\{r, \Sigma\}$*-Problem, we have* $\rho_\Sigma \leq 3$, ρ_Δ *unbounded,* $\rho_{\min}$ *unbounded and* $\rho_{\max} \leq 4$. *The bounds are tight.*

PROOF. Let $f = (x, y)$ be a best swap edge and let $f' = (x', y')$ be the (only) swap edge such that $f' \in E(S'_e(r))$ and f' is on a shortest path from r to v in $S'_e(r)$. Concerning ρ_Σ, let $\overline{\ell_{e/f}}$ and $\overline{\ell'}$ denote the average length of a path from r to $t \in V(S_v)$ in $S_{e/f}(r)$ and $S'_e(r)$, respectively. Of course, $\rho_\Sigma = \overline{\ell_{e/f}}/\overline{\ell'}$. We have

$$\overline{\ell_{e/f}} \leq \overline{\ell_{e/f'}} \leq d_{e/f'}(r, y') + d_{e/f'}(y', v) + \frac{\sum_{t \in V(S_v)} d(v, t)}{|S_v|}$$

and given that $d_{e/f'}(r, y') = d'(r, y')$ and $d_{e/f'}(y', v) = d(y', v) = d'(y', v)$ (since the old path from v to y' was a shortest path), it follows that

$$\overline{\ell_{e/f}} \leq d'(r, v) + \frac{\sum_{t \in V(S_v)} d(v, t)}{|S_v|}.$$

Moreover, we have that for any $t \in V(S_v)$,

$$d'(r, v) \leq d'(r, t) + d(t, v) \leq d'(r, t) + d(r, t) \leq 2d'(r, t)$$

from which, for any node t in S_v, it follows that $d'(r, v) \leq 2\overline{\ell'}$. Furthermore,

$$\frac{\sum_{t \in V(S_v)} d(v, t)}{|S_v|} \leq \frac{\sum_{t \in V(S_v)} d(r, t)}{|S_v|} \leq \frac{\sum_{t \in V(S_v)} d'(r, t)}{|S_v|} = \overline{\ell'}.$$

Therefore, we have that $\overline{\ell_{e/f}} \leq 3\overline{\ell'}$, that is, $\rho_\Sigma \leq 3$. The bound is tight as shown in Figure 3(a).

Concerning ρ_Δ and $\rho_{\min}$, they are unbounded, as shown in Figure 3(b).

Finally, concerning $\rho_{\max}$, we have $\ell_{e/f} \leq d_{e/f}(r, y) + 2h(S_v) \leq d_{e/f}(r, y) + 2\ell'$. Moreover, $d_{e/f}(r, y) \leq d_{e/f'}(r, y) \leq d_{e/f'}(r, v) + d_{e/f'}(v, y)$, and given that $d_{e/f'}(r, v) = d_{e/f'}(r, y') + d_{e/f'}(y', v) = d'(r, v)$ (since f' is on a shortest path from r to v in $S'_e(r)$,

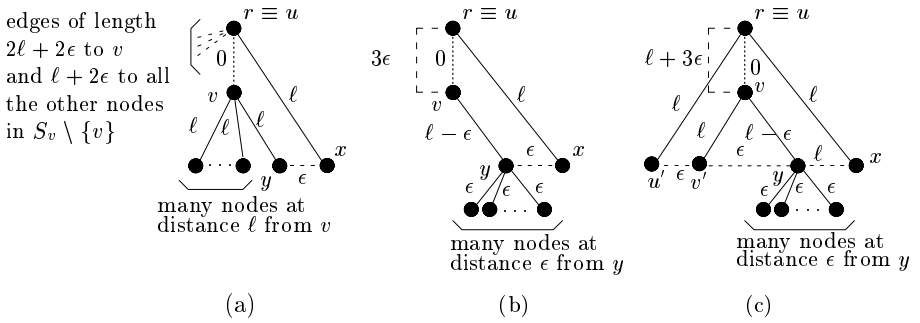

Fig. 3. Ratios for the swap algorithm solving the $\{r, \Sigma\}$-Problem. All parts: $S(r)$ (solid edges) with the removed edge (u, v); nontree edges are dashed and the best swap edge is $f = (x, y)$. (a) In $S_{e/f}(r)$, $\overline{\ell_{e/f}} \to 3\ell + \varepsilon$, while $\overline{\ell'} \to \ell + 2\varepsilon$, $\varepsilon \geq 0$, since in $S'_e(r)$ we have all the edges of length $\ell + 2\varepsilon$ from r to $S_v \setminus \{v, y\}$; then $\rho_\Sigma \to 3$ for $\varepsilon \to 0$. (b) The distance to node v in $S_{e/f}(r)$ is 2ℓ, while in $S'_e(r)$ it is 3ε, $\varepsilon \geq 0$, from which ρ_Δ and $\rho_{\min}$ are unbounded. (c) The distance to node v' in $S_{e/f}(r)$ (i.e., $\ell_{e/f}$) is $4\ell - \varepsilon$, while $\ell' = \ell + 3\varepsilon$, $\varepsilon \geq 0$, from which $\rho_{\max} \to 4$ for $\varepsilon \to 0$.

and this path contains the old shortest path from v to y'), it follows that

$$d_{e/f}(r, y) \leq d'(r, v) + d(v, y) \leq \ell' + h(S_v) \leq 2\ell',$$

that is, $\ell_{e/f} \leq 4\ell'$ or $\rho_{\max} \leq 4$. The bound is tight as shown in Figure 3(c). □

4.2. *Ratios for the Swap Algorithm Solving the* $\{r, \Delta\}$*-Problem.* We can prove the following result:

THEOREM 4.2. *For the swap algorithm solving the* $\{r, \Delta\}$*-Problem, we have* $\rho_\Sigma \leq 3$, $\rho_\Delta = 1$, $\rho_{\min} \leq 2$ *and* $\rho_{\max} \leq 2$. *The bounds are tight.*

PROOF. Let f be a best swap edge and let t be any node in S_v. Concerning ρ_Σ, from the fact that the $\{r, \Delta\}$-Problem reduces to the $\{r, v\}$-Problem, it follows that f is on a shortest path in $G - e$ from r to v. Hence, $S_{e/f}(r)$ and $S'_e(r)$ share that path, and then $d_{e/f}(r, v) = d'(r, v)$. From this, we have that

$$d_{e/f}(r, t) \leq d_{e/f}(r, v) + d(v, t) = d'(r, v) + d(v, t).$$

Since $d'(r, v) \leq d'(r, t) + d(v, t)$ and $d(v, t) \leq d(r, t) \leq d'(r, t)$, it follows that $d_{e/f}(r, t) \leq 3d'(r, t)$, that is, $\rho_\Sigma \leq 3$. The bound is tight as shown in Figure 4(a).

Concerning ρ_Δ, since $d_{e/f}(r, v) = d'(r, v)$, it follows that $\rho_\Delta = 1$.

Concerning $\rho_{\min}$, let $t \in V(S_v)$ be a closest node to r in $G - e$. We have

$$d_{e/f}(r, y) + d(y, v) \leq d'(r, t) + d(t, v)$$

from which

$$d_{e/f}(r, y) \leq d'(r, t) + d(t, v) - d(y, v) \leq d'(r, t) + d(r, t) \leq 2d'(r, t)$$

and then $\rho_{\min} \leq 2$. The bound is tight as shown in Figure 4(b).

Finally, concerning $\rho_{\max}$, we have that $\ell_{e/f} \leq d_{e/f}(r, v) + h(S_v) = d'(r, v) + h(S_v) \leq \ell' + \ell' = 2\ell'$, that is, $\rho_{\max} \leq 2$. The bound is tight as shown in Figure 4(c). □

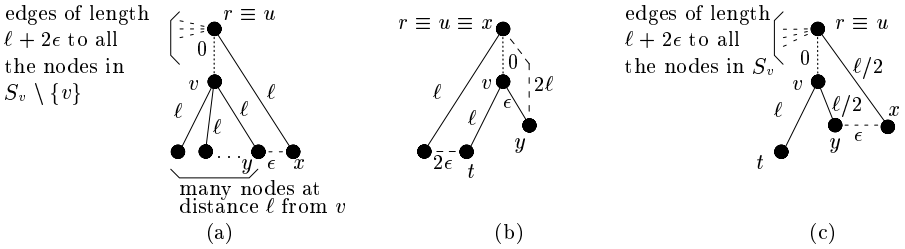

Fig. 4. Ratios for the swap algorithm solving the $\{r, \Delta\}$-Problem. All parts: $S(r)$ (solid edges) with the removed edge (u, v); nontree edges are dashed and the best swap edge is $f = (x, y)$. (a) $\ell_{e/f} \to 3\ell + \varepsilon$, while $\ell' \to \ell + 2\varepsilon$, $\varepsilon \geq 0$, since in $S'_e(r)$ we have all the edges of length $\ell + 2\varepsilon$ from r to $S_v \setminus \{v\}$, from which $\rho_\Sigma \to 3$ for $\varepsilon \to 0$. (b) The closest node to the root in $S_{e/f}(r)$ is y, and $d_{e/f}(r, y) = 2\ell + \varepsilon$, while in $S'_e(r)$ the distance from the root to the closest node t is $d'(r, t) = \ell + 2\varepsilon$, $\varepsilon \geq 0$, from which $\rho_{\min} \to 2$ for $\varepsilon \to 0$. (c) The distance to node t in $S_{e/f}(r)$ (i.e., $\ell_{e/f}$) is $2\ell + \varepsilon$, while $\ell' = \ell + 2\varepsilon$, $\varepsilon \geq 0$, from which $\rho_{\max} \to 2$ for $\varepsilon \to 0$.

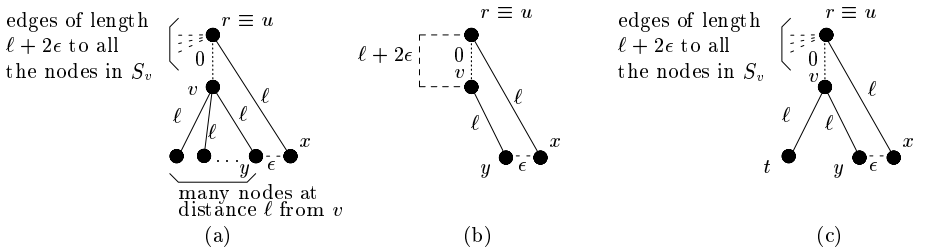

Fig. 5. Ratios for the swap algorithm solving the $\{r, \min\}$-Problem. All parts: $S(r)$ (solid edges) with the removed edge (u, v); nontree edges are dashed and the best swap edge is $f = (x, y)$. (a) $\overline{\ell_{e/f}} \to 3\ell + \varepsilon$, while $\overline{\ell'} \to \ell + 2\varepsilon$, $\varepsilon \geq 0$, since in $S'_e(r)$ we have all the edges of length $\ell + 2\varepsilon$ from r to S_v, from which $\rho_\Sigma \to 3$ for $\varepsilon \to 0$. (b) The distance to node v in $S_{e/f}(r)$ is $2\ell + \varepsilon$, while in $S'_e(r)$ it is $\ell + 2\varepsilon$, $\varepsilon \geq 0$, from which $\rho_\Delta \to 2$ for $\varepsilon \to 0$. (c) The distance to node t in $S_{e/f}(r)$ (i.e., $\ell_{e/f}$) is $3\ell + \varepsilon$, while $\ell' = \ell + 2\varepsilon$, $\varepsilon \geq 0$, from which $\rho_{\max} \to 2$ for $\varepsilon \to 0$.

4.3. *Ratios for the Swap Algorithm Solving the* $\{r, \min\}$*-Problem.* We can prove the following result:

THEOREM 4.3. *For the swap algorithm solving the* $\{r, \min\}$*-Problem, we have* $\rho_\Sigma \leq 3$, $\rho_\Delta \leq 3$, $\rho_{\min} = 1$ *and* $\rho_{\max} \leq 2$. *The bounds are tight.*

PROOF. Note that $f = (x, y)$ belongs to both the trees, since y is the node of S_v closest to r. Therefore, trivially $\rho_{\min} = 1$.

Concerning ρ_Σ and $\rho_{\max}$, let t be any node in S_v. Let z be the nearest common ancestor of y and t in S_v. We have that

$$d_{e/f}(r, t) = d_{e/f}(r, y) + d(y, z) + d(z, t) \leq d'(r, y) + d(y, v) + d(v, t)$$

and given that $d'(r, y) \leq d'(r, t)$, since y is the node of S_v closest to r, and $d(y, v) \leq d(r, y) \leq d'(r, y) \leq d'(r, t)$, and $d(v, t) \leq d'(r, t)$, it follows that $d_{e/f}(r, t) \leq 3d'(r, t)$. Therefore, we have $\rho_\Sigma \leq 3$ and $\rho_{\max} \leq 3$. The bounds are tight as shown in Figure 5(a) and 5(c).

Concerning ρ_Δ, since

$$\begin{aligned} d_{e/f}(r, v) &= d_{e/f}(r, y) + d(y, v) = d'(r, y) + d(y, v) \\ &\leq d'(r, v) + d(r, y) \leq d'(r, v) + d'(r, y) \leq 2d'(r, v) \end{aligned}$$

it follows that $\rho_\Delta \leq 2$. The bound is tight as shown in Figure 5(b). □

4.4. *Ratios for the Swap Algorithm Solving the* $\{r, \max\}$*-Problem.* We can prove the following result:

THEOREM 4.4. *For the swap algorithm solving the* $\{r, \max\}$*-Problem, we have* ρ_Σ *unbounded,* ρ_Δ *unbounded,* $\rho_{\min}$ *unbounded and* $\rho_{\max} \leq 2$. *The bounds are tight.*

PROOF. Let f and f' be defined as for Theorem 4.1. Concerning ρ_Σ, ρ_Δ and $\rho_{\min}$, they are unbounded as shown in Figure 6(a).

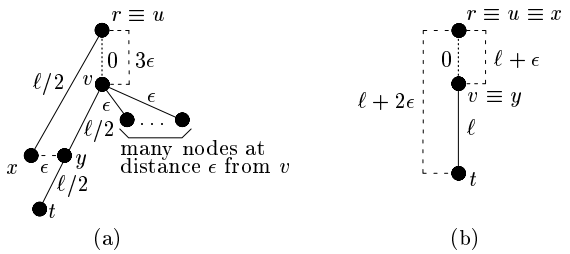

Fig. 6. Ratios for the swap algorithm solving the $\{r, \max\}$-Problem. All parts: $S(r)$ (solid edges) with the removed edge (u, v); nontree edges are dashed and the best swap edge is $f = (x, y)$. (a) $\overline{\ell_{e/f}} \to \ell + 2\varepsilon$, while $\overline{\ell'} \to 4\varepsilon$, and the distance to node v in $S_{e/f}(r)$ is $\ell + \varepsilon$, while in $S'_e(r)$ it is 3ε, $\varepsilon \geq 0$, from which ρ_Σ, ρ_Δ and $\rho_{\min}$ are unbounded. (b) The distance to node t in $S_{e/f}(r)$ (i.e., $\ell_{e/f}$) is $2\ell + \varepsilon$, while in $S'_e(r)$ the height is $\ell' = \ell + 2\varepsilon$, $\varepsilon \geq 0$, from which $\rho_{\max} \to 2$ for $\varepsilon \to 0$.

Concerning $\rho_{\max}$, we have

$$\ell_{e/f} \leq \ell_{e/f'} \leq d_{e/f'}(r, v) + h(S_v) = d'(r, v) + h(S_v) \leq \ell' + h(S_v) \leq 2\ell'$$

from which $\rho_{\max} \leq 2$. The bound is tight as shown in Figure 6(b). □

5. Concluding Remarks

5.1. *Summary of the Obtained Results.* In this paper we have introduced the notion of best swap edge for a failing edge of an SPT, and we have presented several efficient algorithms for computing all the best swap edges of an SPT. These algorithms are very effective, in the sense that the swap tree they create is functionally very close to $S'_e(r)$; moreover, computing a swap tree is much faster than rebuilding from scratch a new SPT $S'_e(r)$ of $G - e$, for each edge $e \in E_S$. Table 1 summarizes the bounds and the ratios of the various algorithms for which we have performed comparisons between $S_{e/f}(r)$ and $S'_e(r)$.

Interestingly, the swap algorithm for the $\{r, \Delta\}$-Problem, which is the cheapest in terms of running time, is also the best with respect to the measures of quality we have defined. Our interpretation is that choosing as a best swap edge that one belonging to

Table 1. Running time, space requirements and ratios for the studied swap algorithms.

	Algorithm			
Measure	$\{r, \Sigma\}$	$\{r, \Delta\}$	$\{r, \min\}$	$\{r, \max\}$
Time	$O(n^2)$	$O(m \cdot \alpha(m, n))$	$O(m \cdot \alpha(m, n))$	$O(n\sqrt{m})$
Space	$O(n^2)$	$O(m)$	$O(m)$	$O(m)$
ρ_Σ	3	3	3	Unbounded
ρ_Δ	Unbounded	1	2	Unbounded
$\rho_{\min}$	Unbounded	2	1	Unbounded
$\rho_{\max}$	4	2	3	2

a new shortest path from r to v (as the swap algorithm for the $\{r, \Delta\}$-Problem does) produces a swap tree topologically and then functionally similar to the old SPT.

It is impressive to note that using such a swap algorithm will save an $O(\max\{n/\alpha(m, n), n^2 \log n/(m \cdot \alpha(m, n))\})$ time factor with respect to recomputing from scratch $S'_e(r)$ for each edge $e \in E_S$, while the most important features of $S_{e/f}(r)$ and $S'_e(r)$ are extremely similar.

5.2. *Possible Extensions*. A large amount of work concerning swap problems in SPTs remains to be done. For example, objective functions others than those we considered in this paper could be defined. As interesting examples in this direction, we mention the class of *many-to-one swap problems*, arising when the function focuses on a subset of nodes in S_u and a fixed node in S_v, and the class of *many-to-many swap problems*, arising when the function focuses on a subset of nodes in S_u and on a subset of nodes in S_v. These classes of problems are quite different from those analyzed in this paper, since they focus on a subset of nodes in S_u, and therefore they de-emphasize the importance of nodes r and u. However, there can be practical situations in which it makes sense to study the above classes of problems.

For example, imagine a scenario in which a message sent from the root r to a destination node t is stored in each node it passes through. Then, as soon as an edge $e = (u, v)$ along the path from r to t fails, we have that all the nodes along the path $P = \langle r, \ldots, u \rangle$ in $S(r)$ can send the same message to v, and we are interested in doing that as fast as possible. The above situation, for $e = (u, v) \in E_S$ and $e' \in \mathcal{C}_e$, can be modeled by the following function:

$$F_{\{P,v\}}(S_{e/e'}(r)) = \min\{d_{e/e'}(s, v): s \in V(P)\}.$$

Hence, this problem belongs to the class of many-to-one swap problems, and it can be shown that it is solvable in $O(m \cdot \alpha(m, n))$ time and $O(m)$ space.

Similarly, interesting examples of many-to-many problems can be obtained by combining many-to-one and one-to-many problems.

5.3. *Open Problems*. Among the problems left open, we mention the case of transient node failures and that of multiple simultaneous link failures. Moreover, it would be very interesting to study the average values of the ratios we have considered. Clearly, it also remains to establish whether the algorithms we have proposed are optimal. All the algorithms making use of a transmuter will need time $\Omega(m \cdot \alpha(m, n))$, the size of the transmuter [12]; to improve beyond that, a different approach must be used. Finally, swap problems in other network topologies (e.g., bipartite networks, spanners, etc.) deserve further study.

Acknowledgments. The authors thank the anonymous referees for the useful suggestions that helped improve the presentation of the paper.

Appendix. In this Appendix we present the proofs for the swap problems in which the node u takes the place of r.

PROOF OF THEOREM 3.2. The proof is similar to that of Theorem 3.1. We start by computing in $O(\alpha(m, n))$ amortized time the nearest common ancestor $z_{e'}$ in $S(r)$ of u' and v', with $e' = (u', v') \in \mathcal{C}_e$ [6]. Clearly, $z_{e'}$ belong to the path in $S(r)$ from r to u. To associate e with its best swap edge, we have to minimize the value of $d_{e/e'}(u, v)$, $\forall e' \in \mathcal{C}_e$. It follows that a best swap edge will be an edge $f = (x, y)$ such that

$$(6)\quad d(u, z_f) + d(z_f, x) + |f| + d(v, y) = \min\{d(u, z_{e'}) + d(z_{e'}, u') + |e'| + d(v, v')\colon\ e' = (u', v') \in \mathcal{C}_e\},$$

which depends on (u, v). To avoid this problem, we associate with e' the length of the cycle in $S(r)$ passing through e', that is,

$$c(e') = d(u, z_{e'}) + d(z_{e'}, u') + |e'| + d(u, v').$$

In fact, $d_{e/e'}(u, v) = c(e') - |e|$ for any edge $e' \in \mathcal{C}_e$ and therefore, with a shortest cycle is associated a best swap edge, and vice versa. Since the above distances can be computed in $O(1)$ time, we have that $c(e')$ can be computed in $O(1)$ time. Finally, we can solve the $\{u, v\}$-Problem by processing the nodes of the transmuter in reverse topological order. This completes the proof. □

PROOF OF THEOREM 3.7. The proof is given point by point.

1. $\{u, \Sigma\}$-Problem. The algorithm is similar to that used for solving the $\{r, \Sigma\}$-Problem. The only exception is that (2) is now transformed as follows, where $z_{e'}$ denotes the nearest common ancestor of u' and t in $S(r)$:

$$\mathit{min\_path}(t, v) = \min\{d(u, z_{e'}) + d(z_{e'}, u') + |e'|\colon\ e' = (u', t) \in \mathcal{C}_e\}.$$

2. $\{u, \Delta\}$-Problem. The proof is similar to that of Theorem 3.4. In fact, also in this case it is easy to see that

$$\max\{d_{e/f}(u, t) - d(u, t)\colon\ t \in V(S_v)\} = \min\{d_{e/e'}(u, v)\colon\ e' \in \mathcal{C}_e\}.$$

Hence, the $\{u, \Delta\}$-Problem reduces to the $\{u, v\}$-Problem, and therefore it can be solved using the approach described in Theorem 3.2. This completes the proof.

3. $\{u, \min\}$-Problem. The proof is similar to to that of Theorem 3.5. For any swap edge $e' = (u', v')$, let $z_{e'}$ denote the nearest common ancestor of u' and v' in $S(r)$. To associate e with its best swap edge f, we have to minimize the value of $d_{e/e'}(u, v')$, $\forall e' \in \mathcal{C}_e$. It follows that a best swap edge will be an edge $f = (x, y)$ such that

$$(7)\quad d(u, x) + |f| = \min\{d(u, u') + |e'|\colon\ e' = (u', v') \in \mathcal{C}_e\},$$

which depends on (u, v). As for the $\{r, \min\}$-Problem, we initially transform G to the graph G' there introduced. Then we build a transmuter $D_{G'}(S(r))$ in which a sink node associated with a nontree edge $e' = (u', v')$, such that $z_{e'} \equiv u'$, is labeled with

$$c(e') = d(u', v') + |e'| - d(r, u'),$$

not depending on (u, v). In fact, $d_{e/e'}(u, v') = c(e') + d(u, u') + d(r, u') = c(e') + d(r, u)$ for any edge $e' \in \mathcal{C}_e$ (now considered in G'). Finally, we can solve the $\{u, \min\}$-Problem

by processing the nodes of the transmuter in reverse topological order. This completes the proof.

4. $\{u, \max\}$-Problem. The proof is similar to that of Theorem 3.6, except that for any swap edge $e' = (u', v')$, we now add to the length of a longest path starting from v' and staying within S_v the length $d_{e/e'}(u, v') = d(u, z_{e'}) + d(z_{e'}, u') + |e'|$, where as usual $z_{e'}$ denotes the nearest common ancestor of u' and v'. □

References

[1] 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.

[2] M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. Assoc. Comput. Mach.*, **34**(3) (1987), 596–615.

[3] D. Frigioni, A. Marchetti-Spaccamela and U. Nanni, Fully dynamic output bounded single source shortest path problem, *Proc. 7th ACM–SIAM Symposium on Discrete Algorithms* (*SODA '96*), 1996, pp. 212–221.

[4] M. Grötschel, C.L. Monma and M. Stoer, Design of survivable networks, *Handbook in OR and MS*, Vol. 7, Elsevier, Amsterdam, 1995, pp. 617–672.

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

[6] D. Harel and R.E. Tarjan, Fast algorithms for finding nearest common ancestors, *SIAM J. Comput.*, **13**(2) (1984), 338–355.

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

[8] K. Iwano and N. Katoh, Efficient algorithms for finding the most vital edge of a minimum spanning tree, *Inform. Process. Lett.*, **48**(5) (1993), 211–213.

[9] E. Nardelli, G. Proietti and P. Widmayer, Finding all the best swaps of a minimum diameter spanning tree under transient edge failures, *J. Graph Algorithms Appl.*, **5**(5) (2001), 39–57.

[10] E. Nardelli, G. Proietti and P. Widmayer, How to swap a failing edge of a single source shortest paths tree, *Proc. 5th Annual International Computing and Combinatorics Conference* (*COCOON '99*), Vol. 1627 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1999, pp. 144–153.

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

[12] R.E. Tarjan, Complexity of monotone networks of computing conjunctions, *Ann. Discrete Math.*, **2** (1975), 121–133.

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

[14] R.E. Tarjan, Sensitivity analysis of minimum spanning trees and shortest path trees, *Inform. Process. Lett.*, **14**(1) (1982), 30–33.