How to Swap a Failing Edge of a Single Source Shortest Paths Tree*

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

[1] Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila, Via Vetoio, 67010 L'Aquila, Italy. {nardelli,proietti}@univaq.it.
[2] Ist. di Analisi dei Sistemi e Informatica, CNR, V.le Manzoni 30, 00185 Roma, Italy.
[3] Institut für Theoretische Informatik, ETH Zentrum, 8092 Zürich, Switzerland. widmayer@inf.ethz.ch.

Abstract. In this paper we introduce the notion of *best swap* for a failing edge of a single source shortest paths tree (SPT) $S(r)$ rooted in r in a weighted graph $G = (V, E)$. Given an edge $e \in S(r)$, an edge $e' \in E \setminus \{e\}$ is a *swap edge* if the *swap tree* $S_{e/e'}(r)$ obtained by swapping e with e' in $S(r)$ is a spanning tree of G. A *best swap edge* for a given edge e is a swap edge minimizing some distance functional between r and the set of nodes disconnected from the root after the edge e is removed. A *swap algorithm* with respect to some distance functional computes a best swap edge for every edge in $S(r)$. We show that there exist fast swap algorithms (much faster than recomputing from scratch a new SPT) which also preserve the functionality of the affected SPT.

1 Introduction

Survivability of a communication network means 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], mainly as a consequence of the advent of sparse fiber optic networks. For example, a classic survivability problem is that of maintaining the shortest path between two specified nodes in the network under the assumption that at most k arcs or nodes along the original shortest path are removed [2].

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 likely to react catastrophically to failures, especially of links, since the density of traffic through each link is very high. Therefore, it is important for sparse networks to take survivability into account from the very beginning. Assuming that damaged links can be restored quickly, the likelihood of having failures that overlap in time is quite small. Therefore, it makes sense to study the problem of dealing with the failure of each single link in the network, since we can expect that sooner or later each link will fail. Moreover, for several practical motivations [6], whenever

* This research was partially supported by the CHOROCHRONOS TMR Program of the European Community. The work of the third author was partially supported by grant "Combinatorics and Geometry" of the Swiss National Science Foundation.

T. Asano et al. (Eds.): COCOON'99, LNCS 1627, pp. 144–153, 1999.

a link fails, it is important that the number of replacing edges is small. If the network is a spanning tree, the optimum would be replacing a link with a single link reconnecting the network. The question is: *which is the link that has to be chosen?*

Coping with a failure of a link in a network having the topology of a tree means, on the theoretical side, to define an interesting family of problems on graphs. Let $G = (V, E)$ be a biconnected, undirected graph, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges, with a nonnegative real length $|e|$ associated with each edge $e \in E$. Let n and m denote the number of vertices and the number of edges, respectively. Let $T = (V, E_T)$, with $E_T \subseteq E$ be a spanning tree of G. A *swap edge* for an edge $e = (u, v) \in E_T$ is an edge $e' = (u', v') \in E \setminus \{e\}$ reconnecting the two subtrees created by the removal of e. Let in the following $\mathcal{S}_e$ denote the set of all swap edges for e and let $T_{e/e'}$ be the tree obtained by swapping e with e'. Let $F[T_{e/e'}]$ be a functional of $T_{e/e'}$. A *best swap* for an edge e is an edge $f \in \mathcal{S}_e$ such that $F[T_{e/f}] \leq F[T_{e/e'}]$, for any $e' \in \mathcal{S}_e$. A *swap algorithm* finds a best swap for every edge $e \in E_T$.

An interesting example arises when T is a minimum spanning tree (MST). Here, the natural functional to be defined is $F[T_{e/e'}] = \sum_{g \in E_{T_{e/e'}}} |g|$. In this case, it is easy to see that f is a swap edge of minimum length. Moreover, $T_{e/f}$ coincides with T', the MST of $G - e = (V, E \setminus \{e\})$. The fastest solution for finding a best swap for each edge in T runs in $O(m \cdot \alpha(m, n))$ time [9], where $\alpha(m, n)$ is the functional inverse of Ackermann's function [8].

Another interesting problem arises when T is a minimum diameter spanning tree (MDST). In this case, the natural functional is the diameter of $T_{e/e'}$, and a best swap is a swap edge which makes the diameter of the new spanning tree as low as possible. The problem of finding a best swap for each edge in T has been solved by Nardelli et al. in $O(n\sqrt{m})$ time [7]. They also showed that the diameter of $T_{e/f}$ is at most 5/2 times the diameter of T', the MDST of $G - e$.

However, many network architectures are based on a single source shortest paths tree (SPT) rooted at a given node r, say $S(r) = (V, E_S)$ [1]. This is especially true for centralized network, where there exists a privileged node broadcasting messages to all the other nodes. In this case, a best swap policy for a failing edge is not unique as for the MST and the MDST, and depends on whatever is of interest from a network management point of view. Therefore, a rigorous approach to the problem requires the definition of the objective functional $F[S_{e/e'}(r)]$ to be minimized, and the complexity of finding a best swap for every edge in $S(r)$ will depend on the selected functional. In this paper, we propose efficient swap algorithms for three functionals of primary importance in SPTs. Let S_v denote the set of nodes disconnected from the root after the removal of $e = (u, v) \in S(r)$ and reconnected via the swap edge e' (see Figure 1).

Moreover, let $d(v_1, v_2)$ (resp., $d_{e/e'}(v_1, v_2)$) denote the distance between v_1 and v_2 in $S(r)$ (resp., $S_{e/e'}(r)$), for any $v_1, v_2 \in V$. The following functionals are considered, due to their primary importance in network applications:

1. $F[S_{e/e'}(r)] = \sum_{t \in S_v} d_{e/e'}(r, t)$;

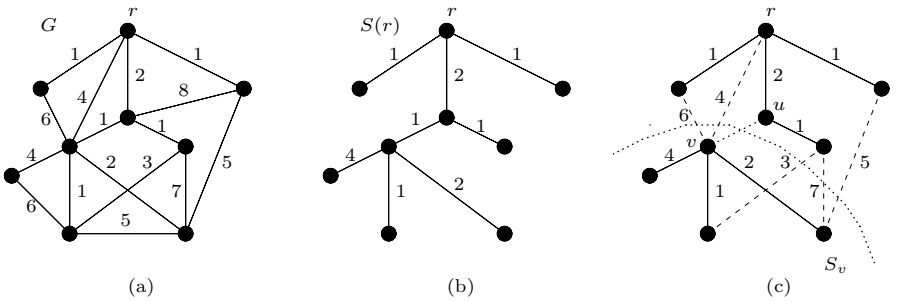

Fig. 1. (a) A weighted graph $G = (V, E)$; (b) a SPT $S(r)$ rooted in r; (c) edge $e = (u, v)$ is removed from $S(r)$: dashed edges are swap edges.

2. $F[S_{e/e'}(r)] = \max_{t \in S_v}\{d_{e/e'}(r, t)\}$;
3. $F[S_{e/e'}(r)] = \max_{t \in S_v}\{d_{e/e'}(r, t) - d(r, t)\}$.

These functionals focus on S_v, since we are interested in studying how the nodes disconnected from the root are affected by the failure. The swap problems induced by these functionals will be named the *sum-problem*, the *height-problem* and the *increase-problem*, respectively. Our analysis shows that there exist fast swap algorithms for these functionals. These algorithms are much faster than recomputing from scratch for every edge $e \in S(r)$ the SPT $S'(r)$ rooted in r of $G - e$, (we say recomputing from scratch since no dynamic solution for this problem is known which is asymptotically better [3]). Moreover, we compare $S_{e/f}(r)$ with $S'(r)$, on the basis of the various studied functionals, showing that with respect to the chosen functionals, $S_{e/f}(r)$ is worse than $S'(r)$ only by a small constant factor, that is, $S_{e/f}(r)$ is *functionally* similar to $S'(r)$, in terms of the studied functionals. Therefore, we conclude that swapping in a SPT is cheap and effective.

The paper is organized as follows: Section 2 proposes the algorithms for solving the problems. In Section 3, we present a comparison between the solutions computed by the various swap algorithms and the respective exact solutions. Finally, in Section 4, we give conclusions and list some open problems.

2 The swap algorithms

2.1 Solving the sum-problem

Remember that the sum-problem asks for a swap edge minimizing the sum of the lengths of all the paths from r to each node in S_v. A brute force solution will consider all the edges in $\mathcal{S}_e$, for any $e \in S(r)$. Since each edge can be a swap edge for $O(n)$ edges, and the above sum can be computed in $O(n)$ time, it follows that such an approach will cost $O(n^2 m)$ time.

We now propose a more efficient solution. A high-level description of our algorithm is the following. We consider all the edges $e = (u, v) \in S(r)$ in any arbitrary postorder. Let us now fix such an edge (u, v). For each node t in S_v we

do the following. We select a swap edge leading from r to t in $G - e$ on a path as short as possible, and we compute the corresponding sum of all the paths starting from r, going to t and leading to all the nodes in S_v. To do so efficiently, during the postorder traversal we keep track of the total length of all paths from t that stay within S_v and of the number of these paths. Finally, we select the minimum over these values for all nodes t in S_v and we return the corresponding best swap edge.

Let us now present a more detailed description of the algorithm. We make use of the following auxiliary data, for any $v \in V$ (see Figure 2):

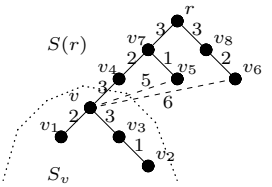

Fig. 2. A weighted graph and a SPT $S(r)$ (solid edges); non-tree edges are dashed. $S_v = \{v, v_1, v_3, v_2\}$, $\texttt{son}(v) = \{v_1, v_3\}$, $size(v) = 4$, $down(v) = 9$, $up(v, v_4) = 3$, $up(v, v_7) = 14$, $min\_path(v, v) = 9$, $min\_path(v, v_4) = 9$, $min\_path(v, v_7) = 11$, $all\_paths(v, v) = 45$, $all\_paths(v, v_4) = 57$, $all\_paths(v, v_7) = 100$.

- $\texttt{son}(v)$: list of sons of v in $S(r)$;
- $size(v)$: number of nodes in S_v;
- $down(v)$: $\sum_{t \in S_v} d(v, t)$;
- $up(v, w)$: $\sum_{t \in S_w \setminus S_v} d(v, t)$, where w is an ancestor of v in $S(r)$, except r;
- $min\_path(v, w)$: $\min_{(e'=(u',v) \in E \setminus E_S) \wedge (u' \notin S_w)} \{d(r, u') + |e'|\}$, where w is either v or an ancestor of v in $S(r)$, except r; if no such edge e' exists, set $min\_path(v, w) = +\infty$;

The algorithm consists of the following steps:

Algorithm SUM_PROBLEM$(G, S(r))$;
Input: A weighted graph $G = (V, E)$ and a SPT $S(r) = (V, E_S)$;
Output: $\forall e = (u, v) \in E_S$, a swap edge $f | \sum_{t \in S_v} d_{e/f}(r, t) = \min_{e' \in \mathcal{S}_e} \left\{ \sum_{t \in S_v} d_{e/e'}(r, t) \right\}$.

`Step 1:` For each node $v \in S(r)$ as considered by any postorder visit
`Step 2:` Compute $\texttt{son}(v), size(v), down(v)$;
`Step 3:` For each ancestor $w \neq r$ of v (including v) compute $min\_path(v, w)$;
`Step 4:` For each edge $e = (u, v) \in S(r)$ as considered by any postorder visit
`Step 5:` For each node $v_i \in \texttt{son}(v)$
`Step 6:` $up(v_i, v) = down(v) - down(v_i) + [size(v) - size(v_i) - 1] \cdot |(v_i, v)|$;
`Step 7:` For each node $t \in S_{v_i}$
`Step 8:` $up(t, v) = up(t, v_i) + [size(v) - size(v_i)] \cdot d(v_i, t) + up(v_i, v)$;
`Step 9:` For each node $t \in S_v$
`Step 10:` $all\_paths(t, v) = down(t) + up(t, v) + min\_path(t, v) \cdot size(v)$;
`Step 11:` Compute $t_{\min}$, where $all\_paths(t_{\min}) = \min_{t \in S_v} \{all\_paths(t, v)\}$;
`Step 12:` Output the edge associated with $min\_path(t_{\min}, v)$.

Theorem 1. *Given a graph $G = (V, E)$ with n vertices and m edges, with positive real edge lengths and a SPT $S(r)$ rooted in $r \in V$, the swap algorithm SUM_PROBLEM(G,S(r)) solves the sum-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 time and space complexity of the algorithm, let us analyze it step by step.

Concerning Step 2, we can compute $\texttt{son}(v)$, $size(v)$ and $down(v)$ in $O(n)$ time and space for each node (and $O(1)$ amortized over all the nodes). Therefore Step 2 can be accomplished in $O(n)$ time and space for all the nodes.

Step 3 can be accomplished in $O(n)$ time and space for each node 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 non-tree edges adjacent to v with respect to their nearest common ancestors. This can be done in $O(1)$ time for each edge [5], that is, it will cost $O(n)$ time for each node and $O(m)$ 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 $b(v, w_{k-1})$. In this way we spend $O(n)$ time and space for each node, and $O(n^2)$ 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 7-11 can be executed in $O(n)$ time, and therefore require a total $O(n^2)$ time for all the nodes. Finally, Step 12 costs $O(1)$ time per node. Therefore, the overall time and space complexity is $O(n^2)$. □

2.2 Solving the height-problem

Remember that the height-problem asks for a swap edge minimizing the length of a longest path starting from r and ending in S_v. The following can be proved:

Theorem 2. *Given a graph $G = (V, E)$ with n vertices and m edges, with positive real edge lengths and a SPT $S(r)$ rooted in $r \in V$, there exists a swap algorithm solving the height-problem in $O(n\sqrt{m})$ time and $O(m)$ space.*

Proof. This problem can be solved by slightly modifying the approach used in [7], where the problem of computing all the best swaps 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 $t \in S_v$ 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. □

2.3 Solving the increase-problem

Remember that the increase-problem asks for a swap edge minimizing the maximum increase of the distance from r to any node in S_v. The following can be proved:

Theorem 3. *Given a graph $G = (V, E)$ with n vertices and m edges, with positive real edge lengths and a SPT $S(r)$ rooted in $r \in V$, there exists a swap algorithm solving the increase-problem in $O(m \cdot \alpha(m, n))$ time and space.*

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

$$\max_{t \in S_v} \{d_{e/f}(r,t) - d(r,t)\} = \min_{e'=(u',v') \in \mathcal{S}_e} \left\{ \max_{t \in S_v} \{d_{e/e'}(r,t) - d(r,t)\} \right\}. \quad (1)$$

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

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

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

Then, (1) becomes $d_{e/f}(r,v) - d(r,v) = \min_{e'=(u',v') \in \mathcal{S}_e} \{d_{e/e'}(r,v) - d(r,v)\}$, and therefore

$$d_{e/f}(r,v) = \min_{e'=(u',v') \in \mathcal{S}_e} \{d_{e/e'}(r,v)\}.$$

Hence, to solve the increase-problem it suffices to get a swap edge minimizing the distance from r to v. To do that efficiently, we make use of a *transmuter* [9]. 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 for each tree edge e a source node $s(e)$, and for each non-tree edge e' a sink node $t(e')$, plus a certain 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 $\mathcal{S}_e$ 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 non-tree edges. This can be done in $O(m \cdot \alpha(m, n))$ time and space [9]. To associate e with its best swap f, it remains to establish the value that has to be given to a sink node. If we associate in $O(1)$ time with e' the length of the (not simple) cycle in $S(r)$ starting from r, passing through e' and going back to r, that is

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

we will have $d_{e/e'}(r,v) = c(e') - d(r,v)$ for any edge $e' \in \mathcal{S}_e$ and therefore, a shortest cycle is associated with a best swap, and vice-versa. Finally, we can solve the increase-problem by processing the nodes of the transmuter in reverse topological order in $O(m \cdot \alpha(m, n))$ time. This completes the proof. □

3 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 with a true SPT that does not use the failed edge. In this section we address this task, comparing the SPT $S_{e/f}(r)$, obtained by swapping the failed edge e with a best swap edge f, and a true SPT $S'(r)$ of $G - e$. While it is natural to study each of the three quality criteria (functionals) 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). Let $d'(v_1, v_2)$ denote the distance between v_1 and v_2 in $S'(r)$. For each swap algorithm, the following ratios in the two trees will be studied:

$$\sigma = \frac{\sum_{t \in S_v} d_{e/f}(r,t)}{\sum_{t \in S_v} d'(r,t)}; \qquad \rho = \frac{\max_{t \in S_v}\{d_{e/f}(r,t)\}}{\max_{t \in S_v}\{d'(r,t)\}}; \qquad \delta = \frac{d_{e/f}(r,v)}{d'(r,v)}. \tag{2}$$

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

3.1 Ratios of the swap algorithm for the sum-problem

We can prove the following result:

Theorem 4. *For the swap algorithm solving the sum-problem, we have $\sigma \leq 3$, $\rho \leq 4$ and δ unbounded. 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 $f' \in S'(r)$ and f' is on the shortest path from r to v in $S'(r)$. Concerning σ, let $\overline{\ell_{e/f}}$ and $\overline{\ell'}$ denote the average length of a path from r to $t \in S_v$ in $S_{e/f}(r)$ and $S'(r)$, respectively. Of course, $\sigma = \overline{\ell_{e/f}}/\overline{\ell'}$. We have

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

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

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

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

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

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

Concerning ρ, 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 the shortest path from r to v in $S'(r)$, and this path contains the old shortest path from v to y') , it follows

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

that is, $\ell_{e/f} \leq 4\ell'$ or $\rho \leq 4$. The bound is tight as shown in Figure 3b.

Finally, concerning δ, it is unbounded as shown in Figure 3c. □

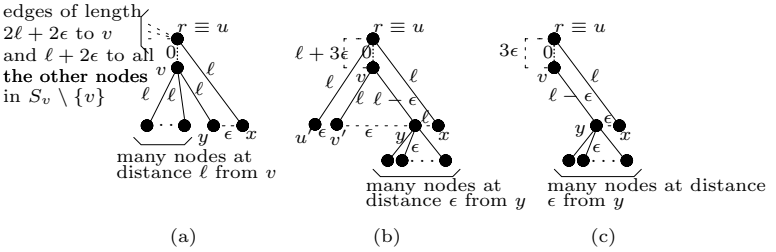

Fig. 3. In all the pictures, $S(r)$ (solid edges) with the removed edge (u,v); non-tree edges are dashed and the best swap is $f = (x,y)$. (a) In $S_{e/f}(r)$, $\overline{\ell_{e/f}} = 3\ell + \epsilon$, while $\overline{\ell'} = \ell + 2\epsilon, \epsilon \geq 0$, since in $S'(r)$ we have all the edges of length $\ell + 2\epsilon$ from r to $S_v \setminus \{v,y\}$; then, $\sigma = 3$. (b) The distance to node v' in $S_{e/f}(r)$ (i.e., the height) is $4\ell - \epsilon$, while in $S'(r)$ the height is $\ell + 3\epsilon, \epsilon \geq 0$, from which $\rho = 4$. (c) The distance to node v in $S_{e/f}(r)$ is 2ℓ, while in $S'(r)$ is $3\epsilon, \epsilon \geq 0$, from which δ is unbounded.

3.2 Ratios of the swap algorithm for the height-problem

We can prove the following result:

Theorem 5. *For the swap algorithm solving the height-problem, we have σ unbounded, $\rho \leq 2$ and δ unbounded. The bounds are tight.*

Proof. Let f and f' be defined as for Theorem 4. Concerning σ and δ, they are unbounded as shown in Figure 4a. Concerning ρ, 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 $\rho \leq 2$. The bound is tight as shown in Figure 4b. □

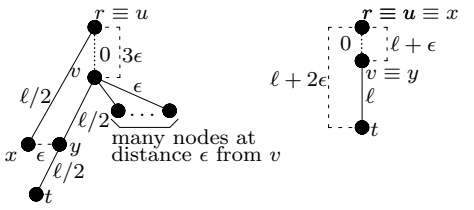

Fig. 4. In all the pictures, $S(r)$ (solid edges) with the removed edge (u, v); non-tree edges are dashed and the best swap is $f = (x, y)$. (a) $\overline{\ell_{e/f}} = \ell + 2\epsilon$, while $\overline{\ell'} = 4\epsilon$, and the distance to node v in $S_{e/f}(r)$ is $\ell + \epsilon$, while in $S'(r)$ is $3\epsilon, \epsilon \geq 0$, from which σ and δ are unbounded. (b) The distance to node t in $S_{e/f}(r)$ (i.e., the height) is $2\ell + \epsilon$, while in $S'(r)$ the height is $\ell + 2\epsilon, \epsilon \geq 0$, from which $\rho = 2$.

3.3 Ratios of the swap algorithm for the increase-problem

We can prove the following result:

Theorem 6. *For the swap algorithm solving the increase-problem, we have* $\sigma \leq 3$, $\rho \leq 2$ *and* $\delta = 1$. *The bounds are tight.*

Proof. Let f be a best swap edge and let t be any node in S_v. Concerning σ, from the fact that f is on the shortest path in $G - e$ from r to v, and then $S_{e/f}(r)$ and $S'(r)$ share that path, we have that $d_{e/f}(r, t) \leq d_{e/f}(r, v) + d(v, t) = d'(r, v) + d(v, t)$ and with $d'(r, v) \leq d'(r, t) + d(v, t)$ 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 $\sigma \leq 3$. The bound is tight as shown in Figure 5a.

Concerning ρ, 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 $\rho \leq 2$. The bound is tight as shown in Figure 5b.

Finally, concerning δ, since $d_{e/f}(r, v) = d'(r, v)$, it follows that $\delta = 1$. □

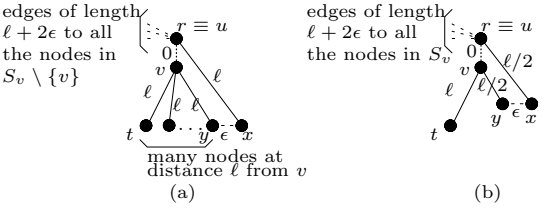

Fig. 5. In all the pictures, $S(r)$ (solid edges) with the removed edge (u, v); non-tree edges are dashed and the best swap is $f = (x, y)$. (a) $\overline{\ell_{e/f}} = 3\ell + \epsilon$, while $\overline{\ell'} = \ell + 2\epsilon, \epsilon \geq 0$, since in $S'(r)$ we have all the edges of length $\ell + 2\epsilon$ from r to $S_v \setminus \{v\}$, from which $\sigma = 3$. (b) The distance to node t in $S_{e/f}(r)$ (i.e., the height) is $2\ell + \epsilon$, while in $S'(r)$ the distance to all the nodes is $\ell + 2\epsilon, \epsilon \geq 0$, from which $\rho = 2$.

4 Summary and conclusions

Table 1 summarizes the bounds of the various algorithms for which we have performed comparisons between $S_{e/f}(r)$ and $S'(r)$. Interestingly, the algorithm

for the increase-problem, which is the cheapest in terms of time complexity, 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 the new shortest path from r to v (as does the swap algorithm for the increase-problem) produces a swap tree topologically similar to the old SPT.

	Algorithm		
Measure	**sum-problem**	**height-problem**	**increase-problem**
Time	$O(n^2)$	$O(n\sqrt{m})$	$O(m \cdot \alpha(m,n))$
Space	$O(n^2)$	$O(m)$	$O(m \cdot \alpha(m,n))$
σ	3	unbounded	3
ρ	4	2	2
δ	unbounded	unbounded	1

Table 1. Time and space complexity and ratios for the studied swap algorithms

After we introduced in this paper the notion of *best swap edge* for a SPT, a large amount of work remains to be done. For example, different functionals $F[S_{e/e'}(r)]$ could be defined. Another open problem is the case of managing multiple simultaneous link failures. Clearly, it also remains to establish whether the algorithms we have proposed are optimal or not. Finally, the case of transient node failures deserves further studies.

Acknowledgements – The authors would like to thank Samir Khuller for helpful discussions on the topic.

References

1. R.K. Ahuia, T.L. Magnanti and J.B Orlin. *Network Flows: Theory, Algorithms and Applications*, Prentice Hall, Englewood Cliffs, NJ (1993).
2. A. Bar-Noy, S. Khuller and B. Schieber, The complexity of finding most vital arcs and nodes, CS-TR-3539, Dept. of Computer Science, Univ. of Maryland, 1995.
3. D. Frigioni, A. Marchetti-Spaccamela and U. Nanni, Fully dynamic output bounded single source shortest path problem, *in Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA'96)*, 1996, 212–221.
4. M. Grötschel, C.L. Monma and M. Stoer, Design of survivable networks, in: *Handbooks in OR and MS, Vol. 7*, Elsevier (1995) 617–672.
5. D. Harel and R.E. Tarjan, Fast algorithms for finding nearest common ancestors, *SIAM J. Comput.*, **13**(2) (1984) 338–355.
6. G.F. Italiano and R. Ramaswami, Maintaining spanning trees of small diameter, *Proc. 21st Int. Coll. on Automata, Languages and Programming (ICALP'94)*, 1994, Lecture Notes in Computer Science, Vol. 820, 212–223.
7. E. Nardelli, G. Proietti and P.Widmayer, Finding all the best swaps of a minimum diameter spanning tree under transient edge failures, *Proc. 6th European Symp. on Algorithms (ESA'98)*, 1998, Lecture Notes in Computer Science, Vol. 1461, 55–66.
8. R.E. Tarjan, Efficiency of a good but not linear set union algorithm, *Journal of the ACM*, **22** (1975) 215–225.
9. R.E. Tarjan, Applications of path compression on balanced trees, *Journal of the ACM*, **26** (1979) 690–715.