Effective Edge-Fault-Tolerant Single-Source Spanners via Best (or Good) Swap Edges

Davide Bilò[1], Feliciano Colella[2], Luciano Gualà[3], Stefano Leucci[4], and Guido Proietti[5,6]

[1] Dipartimento di Scienze Umanistiche e Sociali, University of Sassari, Italy
davide.bilo@uniss.it
[2] Gran Sasso Science Institute, L'Aquila, Italy. feliciano.colella@gssi.it
[3] Dipartimento di Ingegneria dell'Impresa, University of Rome "Tor Vergata", Italy.
guala@mat.uniroma2.it
[4] Department of Computer Science, ETH Zürich, Switzerland.
stefano.leucci@inf.ethz.ch
[5] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, University of L'Aquila, Italy. guido.proietti@univaq.it
[6] Istituto di Analisi dei Sistemi ed Informatica, CNR, Rome, Italy.

Abstract. Computing *all best swap edges* (ABSE) of a spanning tree T of a given n-vertex and m-edge undirected and weighted graph G means to select, for each edge e of T, a corresponding non-tree edge f, in such a way that the tree obtained by replacing e with f enjoys some optimality criterion (which is naturally defined according to some objective function originally addressed by T). Solving efficiently an ABSE problem is by now a classic algorithmic issue, since it conveys a very successful way of coping with a (transient) *edge failure* in tree-based communication networks: just replace the failing edge with its respective swap edge, so as that the connectivity is promptly reestablished by minimizing the rerouting and set-up costs. In this paper, we solve the ABSE problem for the case in which T is a *single-source shortest-path tree* of G, and our two selected swap criteria aim to minimize either the *maximum* or the *average stretch* in the swap tree of all the paths emanating from the source. Having these criteria in mind, the obtained structures can then be reviewed as *edge-fault-tolerant single-source spanners*. For them, we propose two efficient algorithms running in $O(mn + n^2 \log n)$ and $O(mn \log \alpha(m, n))$ time, respectively, and we show that the guaranteed (either maximum or average, respectively) stretch factor is equal to 3, and this is tight. Moreover, for the maximum stretch, we also propose an almost linear $O(m \log \alpha(m, n))$ time algorithm computing a set of *good* swap edges, each of which will guarantee a relative approximation factor on the maximum stretch of 3/2 (tight) as opposed to that provided by the corresponding BSE. Surprisingly, no previous results were known for these two very natural swap problems.

1 Introduction

Nowadays there is an increasing demand for an *efficient* and *resilient* information exchange in communication networks. This means to design on one hand a logical structure onto a given communication infrastructure, which optimizes some sought routing protocol in the absence of failures, and on the other hand, to make such a structure resistant against possible link/node malfunctioning, by suitably strengthening it. In order to address this latter requirement, a classic approach is that of adding a set of redundant links to the structure, which will enter into operation as soon as a failure takes place.

In such a scenario, a key role is certainly played by the *Shortest-Path Tree* (SPT), which is largely used for implementing efficiently the *broadcasting* communication primitive. Indeed, let us model the underlying communication network through an n-vertex and m-edge undirected and with positive real edge weights input graph $G = (V(G), E(G), w)$. Then, for a distinguished source vertex $s \in V(G)$, an SPT rooted at s, say T, can be used to construct a routing table at each vertex of G, in which each entry points out an adjacent vertex laying on a shortest path from s to any other vertex in the network. Despite its efficiency, however, an SPT is very sensitive to failures, since already an edge disruption will disconnect it. Thus, as mentioned above, to maintain alive the communication in this undesired scenario, one should make T *edge-fault-tolerant*, by adding to it a (possibly *sparse*) set of *additional* edges.

Making an SPT edge-fault-tolerant by carefully trading off between the size of the set of additional edges and the quality of the resulting paths emanating from s is a challenging task. If one pushes on the requirement of still having an SPT even after an edge failure, then it is not hard to see that $\Theta(n^2)$ additional edges (on dense graphs) may be needed [9]. Notice that, in contrast, on unweighted graphs, $O(n^{3/2})$ edges are sufficient [21]. On the other hand, in [3] the authors showed that for any arbitrary constant $\varepsilon > 0$, a $(1+\varepsilon)$-approximate edge-fault-tolerant SPT with only a slightly superlinear number of edges can be obtained in polynomial time, namely a structure which retains post-failure paths from the source which are *stretched* up to a factor of $(1+\varepsilon)$. By using a different terminology, such a structure is also known as an *edge-fault-tolerant single-source* $(1+\epsilon)$*-spanner*.

But what about the situation in which one insists on the strong constraint of forcing the size of the edge-fault-tolerant SPT to be linear? In this case, at the state of the art, there is no solution which can provide post-failure paths from the source which are stretched by a factor better than 3.[7] This result can be obtained by solving a so-called *all best swap edges* (ABSE) problem on T, namely the following: for each edge $e = (u, v) \in E(T)$, with u closer to s than v, find the edge reconnecting the two subtrees of T induced by the removal of e that minimizes the distance from s to v. Actually, the corresponding 3-stretched

[7] Notice that, quite interestingly, one can instead design a linear-size data structure, a.k.a. *single-source distance sensitivity oracle*, which can return in $O(1)$ time 2-stretched post-failure distances from s [4].

edge-fault-tolerant SPT is then obtained by adding to T exactly this set of best swap edges. As shown in [20], this problem can be solved very efficiently in $O(m\,\alpha(m,n))$ time, where α is the inverse of the Ackermann function. However, it should be pointed out that the swap edges computed by such an algorithm are *not* best possible w.r.t. the swap criterion of minimizing the stretch from s, and so a natural problem is that of devising an efficient algorithm for such criterion.

Previous work. Due to their fault-tolerance application counterpart, ABSE problems received much attention by the algorithmic community. Generally speaking, an ABSE problem can be defined by mixing up the following two ingredients: a spanning tree of G enjoying some optimality criterion, and a swap criterion defining the desirability of a swap edge, in terms of some feature of the resulting *swap tree* (i.e., that obtained by the original spanning tree after a corresponding swap). Quite naturally, to make consistent an ABSE problem, the swap criterion should get along with the optimization criterion used to design the initial spanning tree.

In particular, swapping in an SPT is a well-recognized approach, since it was shown already in [17] that an effective broadcast routing protocol can be put in place just after the original SPT undergoes an edge failure. Thus, several papers have analyzed the problem in various respects. The interesting point is that an SPT enjoys several optimality criteria when looking at distances from the source s: it minimizes the length of any path from s, and so it minimizes the eccentricity of s and the average distance to any node. Consequently, solving the ABSE problem w.r.t. these two latter measures involved several efforts, and the currently fastest solutions run in $O(m \log \alpha(m,n))$ time [6] and $O(m\,\alpha(n,n) \log^2 n)$ time [10], respectively. Other interesting swap criteria which have been analyzed include the minimization of the maximum increase (before and after the failure) of the distance from s, and the minimization of the distance from s to the root of the subtree that gets disconnected after the failure [19]. Besides the centralized setting, all these swap problems have been studied also in a distributed framework (e.g., see [12–14, 8]).

On the other hand, no results are known for the case in which one is willing to select a BSE with the goal of minimizing either the *maximum* or the *average stretch* from the source s. This is very surprising, since the universally accepted criterion for approximately preserving shortest paths in a graph (also in the presence of failures) is that of computing a *spanner*, namely a sparse subgraph whose paths (between pairs of vertices of interest) are stretched at most (or in the average, less interestingly) by a small factor. In this paper, we aim to fill this gap, by providing efficient solutions exactly for these two swap criteria.

Our results. Let us denote by `ABSE-MS` and `ABSE-AS` the ABSE problem w.r.t. the maximum and the average stretch swap criterion, respectively. For such problems, we devise two efficient algorithms running in $O(mn + n^2 \log n)$ and $O(mn \log \alpha(m,n))$ time, respectively. Notice that both solutions incorporate the running time for computing all the replacement shortest paths from the source after the failure of every edge of the SPT, as provided in [15], whose

computation essentially dominates in an asymptotic sense the time complexity. Our two solutions are based on independent ideas, as described in the following:

- for the `ABSE-MS` problem, we develop a *centroid decomposition* of the SPT, and we exploit a distance property that has to be enjoyed by a BSE w.r.t. a nested and log-depth hierarchy of centroids, which will be defined by the subtree detached from the source after the currently analyzed edge failure. A further simple filtering trick on the set of potential swap edges will allow to reduce them from $O(m)$ to $O(n)$, thus returning the promised $O(n^2 \log n)$ time.
- for the `ABSE-AS` problem, we instead suitably combine a set of linearly-computable (at every edge fault) information, that essentially will allow to describe in $O(1)$ time the quality of a swap edge. This procedure is in principle not obvious, since to compute the average stretch we need to know, for each swap edge, the $O(n)$ distances to all the nodes in the detached subtree. Again, by filtering on the set of potential swap edges, we will get an $O(n^2)$ running time, which will be absorbed by the all-replacement paths time complexity.

Concerning the quality of the corresponding swap trees, we instead show that the guaranteed (either maximum or average, respectively) stretch factor w.r.t. the paths emanating from the source (in the surviving graph) is equal to 3, and this is tight. This is the reason why we name them *effective* edge-fault-tolerant single-source spanners: they can be computed quickly, are very sparse, provide a very simple alternative post-failure routing, and finally have a small (either maximum or average) stretch.

Although the proposed solutions are quite efficient, their running time can become prohibitive for large and dense input graphs, since in this case they would amount to a time cubic in the number of vertices. Unfortunately, it turns out that their improvement is unlikely to be achieved, unless one could avoid the explicit recomputation of all post-failure distances from the source. To circumvent this problem, we then adopt a different approach, which by the way finds application for the (most relevant) max-stretch measure only: we renounce to optimality in the detection of a BSE, in return of a substantial improvement (in the order of a linear factor in n) in the runtime. More precisely, for such a measure, we will compute in an almost linear $O(m \log \alpha(m, n))$ time a set of *good* swap edges (GSE), each of which will guarantee a relative approximation factor on the maximum stretch of 3/2 (tight) as opposed to that provided by the corresponding BSE. Moreover, a GSE will still guarantee an absolute maximum stretch factor w.r.t. the paths emanating from the source (in the surviving graph) equal to 3 (tight).

Besides that, we also point out another important feature concerned with the computation in a *distributed* setting of all our good swap edges. Indeed, in [8] it was shown that they can be computed in an *asynchronous message passing*

system in essentially optimal *ideal time*,[8] space usage, and message complexity, as opposed to the recomputation of all the corresponding BSE, for which no efficient solution is currently available.

Other related results. For the sake of completeness, we quickly recall the main results concerned with ABSE problems. For the *minimum spanning tree* (MST), a BSE is of course one minimizing the *cost* of the swap tree, i.e., a swap edge of minimum cost. This problem is also known as the MST *sensitivity analysis* problem, and can be solved in $O(m \log \alpha(m,n))$ time [22]. Concerning the *minimum diameter spanning tree*, a BSE is instead one minimizing the *diameter* of the swap tree [16, 19], and the best solution runs in $O(m \log \alpha(m,n))$ time [6]. Regarding the *minimum routing-cost spanning tree*, a BSE is clearly one minimizing the *all-to-all routing cost* of the swap tree [24], and the fastest solutions for solving this problem has a running time of $O\left(m2^{O(\alpha(n,n))} \log^2 n\right)$ [5]. Finally, for a *tree spanner*, a BSE is one minimizing the maximum stretch w.r.t. the all pair distances, and the fastest solution to date run in $O(m^2 \log \alpha(m,n))$ time [1].

To conclude, we point out that the general problem of designing fault-tolerant spanners for the *all-to-all* case has been extensively studied in the literature, and we refer the interested reader to [7, 11, 2] and the references therein.

2 Problem Definition

Let $G = (V(G), E(G), w)$ be a 2-edge-connected, edge-weighted, and undirected graph with cost function $w : E(G) \to \mathbb{R}^+$. We denote by n and m the number of vertices and edges of G, respectively. If $X \subseteq V$, let $E(X)$ be the set of edges incident to at least one vertex in X. Given an edge $e \in E(G)$, we will denote by $G - e$ the graph obtained from G by removing edge e. Similarly, given a vertex $v \in V(G)$, we will denote by $G - v$ the graph obtained from G by removing vertex v and all its incident edges. Let T be an SPT of G rooted at $s \in V(G)$. Given an edge $e \in E(T)$, we let $C(e)$ be the set of all the *swap edges* for e, i.e., all edges in $E(G) \setminus \{e\}$ whose endpoints lie in two different connected components of $T - e$, and let $C(e, X)$ be the set of all the swap edge for e incident to a vertex in $X \subseteq V(G)$. For any $e \in E(T)$ and $f \in C(e)$, let $T_{e/f}$ denote the *swap tree* obtained from T by replacing e with f. Let $T_v = (V(T_v), E(T_v))$ be the subtree of T rooted at $v \in V(G)$. Given a pair of vertices $u, v \in V(G)$, we denote by $d_G(u, v)$ the *distance* between u and v in G. Moreover, for a swap edge $f = (x, y)$, we assume that the first appearing endvertex is the one closest to the source, and we may denote by $w(x, y)$ its weight. We define the *stretch factor of y w.r.t.* s, T, G as $\sigma_G(T, y) = \frac{d_T(s,y)}{d_G(s,y)}$.

Given an SPT T of G, the `ABSE-MS` problem is that of finding, for each edge $e = (a, b) \in E(T)$, a swap edge f^* such that:

[8] This is the time obtained with the ideal assumption that the communication time of each message to a neighboring process takes constant time, as in the synchronous model.

$$f^* \in \arg\min_{f \in C(e)} \left\{ \mu(f) := \max_{v \in V(T_b)} \sigma_{G-e}\big(T_{e/f}, v\big) \right\}.$$

Similarly, the ABSE-AS problem is that of finding, for each edge $e = (a, b) \in E(T)$, a swap edge f^* such that:

$$f^* \in \arg\min_{f \in C(e)} \left\{ \lambda(f) := \frac{1}{|V(T_b)|} \sum_{v \in V(T_b)} \sigma_{G-e}\big(T_{e/f}, v\big) \right\}.$$

We will call $\mu(f)$ (resp., $\lambda(f)$) the *max*-(resp., *avg*-)*stretch* of f w.r.t. e.

3 An Algorithm for ABSE-MS

In this section we will show an efficient algorithm to solve the ABSE-MS problem in $O(mn + n^2 \log n)$ time. Notice that a brute-force approach would require $O(mn^2)$ time, given by the $O(n)$ time which is needed to evaluate the quality of each of the $O(m)$ swap edges, for each of the $n - 1$ edges of T. Our algorithm will run through $n - 1$ phases, each returning in $O(m + n \log n)$ time a BSE for a failing edge of T, as described in the following.

Let us fix $e = (a, b)$ as the failing edge. First, we compute in $O(m + n \log n)$ time all the distances in $G - e$ from s. Then, we filter the $O(m)$ potential swap edges to $O(n)$, i.e., at most one for each node v in T_b. Such a filtering is simply obtained by selecting, out of all edges $f = (x, v) \in C(e, \{v\})$, the one minimizing the measure $d_G(s, x) + w(f)$. Indeed, it is easy to see that the max-stretch of such selected swap edge is never worse than that of every other swap edge in $C(e)$. This filtering phase will cost $O(m)$ total time. As a consequence, we will henceforth assume that $|C(e)| = O(n)$.

Then, out of the obtained $O(n)$ swap edges for e, we further restrict our attention to a subset of $O(\log n)$ *candidates* as BSE, which are computed as follows. Let Λ denote a generic subtree of T_b, and assume that initially $\Lambda = T_b$. First of all, we compute in $O(|V(\Lambda)|)$ time a *centroid* c of Λ, namely a node whose removal from Λ splits Λ in a forest F of subtrees, each having at most $|V(\Lambda)|/2$ nodes [18]; then, out of all the swap edges, we select a candidate edge f minimizing the distance from s to c in $T_{e/f}$, i.e.,

$$f \in \arg\min_{(x', v') \in C(e)} \big\{ d_T(s, x') + w(x', v') + d_T(v', c) \big\};$$

then, we compute a *critical node* z for the selected swap edge f, i.e.,

$$z \in \arg\max_{z' \in V(T_b)} \sigma_{G-e}\big(T_{e/f}, z'\big).$$

We now select a suitable subtree Λ' of the forest F, and we pass to the selection of the next candidate BSE by recursing on Λ', until $|V(\Lambda')| = 1$. More precisely, Λ' is the first tree of F containing the first vertex of $V(\Lambda)$ that is encountered by following the path in T from z towards c (see Figure 1).

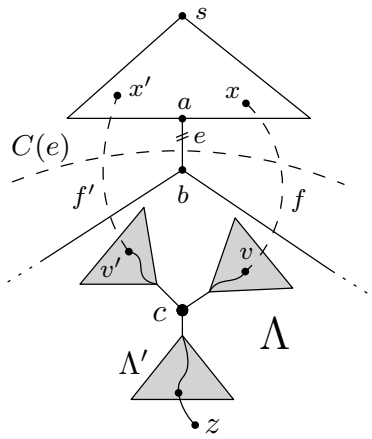

Fig. 1. The situation illustrated in Lemma 1. The subtree Λ is represented by the three gray triangles along with the vertex c. $f = (x, v)$ is the candidate swap edge for e that minimizes $d_T(s, x) + w(f) + d_T(v, c)$, and z is its corresponding critical node. The algorithm will compute the next candidate swap edge by recursing on Λ'.

Due to the property of the centroid, the number of recursions will be $O\big(\log |V(T_b)|\big) = O(\log n)$, as promised, each costing $O(n)$ time. Moreover, at least one of the candidate edges will be a BSE for e, and hence it suffices to choose the edge minimizing the maximum stretch among the corresponding $O(\log n)$ candidate edges. This step is done within the recursive procedure by comparing the current candidate edge f with the best candidate resulting from the nested recursive calls.

A more formal description of each phase is shown in Algorithm 1. In the following we prove the correctness of our algorithm.

Lemma 1. *Let $e = (a, b)$ be a failing edge, and let Λ be a subtree of T_b. Given a vertex $c \in V(\Lambda)$, let $f \in \arg\min_{(x',v') \in C(e)} \big\{d_T(s, v') + w(x', v') + d_T(v', c)\big\}$ and let z be a critical node for f. Let F be the forest obtained by removing the edges incident to c from Λ, and let Λ' be the tree of F containing the first vertex of the path from z to c in T that is also in $V(\Lambda)$. For any swap edge $f' \in C(e, V(\Lambda))$, if $\mu(f') < \mu(f)$ then $f' \in C(e, V(\Lambda'))$.*

Proof. Let $f = (x, v)$ and $f' = (x', v')$. We show that if $v' \in V(\Lambda) \setminus V(\Lambda')$ then $\mu(f') \geq \mu(f)$ (see also Figure 1). Indeed:

$$\begin{aligned}
\mu(f') &\geq \sigma_{G-e}\big(T_{e/f'}, z\big) = \frac{d_{T_{e/f'}}(s, z)}{d_{G-e}(s, z)} = \frac{d_T(s, x') + w(f') + d_T(v', c) + d_T(c, z)}{d_{G-e}(s, z)} \\
&\geq \frac{d_T(s, x) + w(f) + d_T(v, c) + d_T(c, z)}{d_{G-e}(s, z)} \geq \frac{d_{T_e/f}(s, z)}{d_{G-e}(s, z)} = \sigma_{G-e}\big(T_{e/f}, z\big) \\
&= \mu(f),
\end{aligned}$$

Algorithm 1: ABSE-MS(e, Λ)

Input : a failing edge $e = (a, b) \in E(T)$, a subtree Λ of T_b.
Output : an edge $f \in C(e)$. If $C\big(e, V(\Lambda)\big)$ contains a BSE for e, f is a BSE for e.

1 $c \leftarrow$ Centroid of Λ;
2 Let $f \in \arg\min_{(x',v') \in C(e)} \big\{d_T(s, x') + w(x', v') + d_T(v', c)\big\}$;
3 **if** $|V(\Lambda)| = 1$ **then return** f;
4 Let $z \in \arg\max_{z' \in V(T_b)} \sigma_{G-e}\big(T_{e/f}, z'\big)$;
5 Let F be the forest obtained by removing the edges incident to c from Λ;
6 $y \leftarrow$ first vertex along the path from z towards c in T that is also in $V(\Lambda)$;
7 $\Lambda' \leftarrow$ tree of F containing y;
8 $f' \leftarrow$ ABSE-MS(e, Λ');
9 **if** $\mu(f') < \mu(f)$ **then return** f' **else return** f;

where we used the equality $d_T(v', z) = d_T(v', c) + d_T(c, z)$, which follows from the fact that the path from v' to z in T must traverse c as v' and z are in two different trees of F. □

Lemma 2. *If $C\big(e, V(\Lambda)\big)$ contains a BSE for e then ABSE-MS(e, Λ) returns a BSE for e.*

Proof. First of all notice that Algorithm 1 only returns edges in $C(e)$.

We prove the claim by induction on $|V(\Lambda)|$. If $|V(\Lambda)| = 1$ and $C\big(e, V(\Lambda)\big)$ contains a BSE f^* for e, then let f be the edge of $C(e)$ returned by Algorithm 1 and let $V(\Lambda) = \{c\}$. By choice of f, for every $v \in V(T_b)$,

$$d_{T_{e/f}}(s, v) \leq d_{T_{e/f}}(s, c) + d_T(c, v) = d_{T_{e/f^*}}(s, c) + d_T(c, v) = d_{T_{e/f^*}}(s, v),$$

from which we derive that $\mu(f) = \mu(f^*)$, and the claim follows.

If $|V(\Lambda)| > 1$ and $C\big(e, V(\Lambda)\big)$ contains a BSE for e, we distinguish two cases depending on whether the edge f computed by Algorithm 1 is a BSE for e or not. If that is the case, then $\mu(f) \leq \mu(f'') \, \forall f'' \in C(e)$ and the algorithm correctly returns f. Otherwise, by Lemma 1, any edge $f' \in C(e, V(\Lambda))$ such that $\mu(f') < \mu(f)$ must belong to $C\big(e, V(\Lambda')\big)$. It follows that Λ' contains a BSE for e and since $1 \leq |V(\Lambda')| < |V(\Lambda)|$ we have, by inductive hypothesis, that the edge f' returned by ABSE-MS(G, e, Λ') is a BSE for e. Clearly $\mu(f') < \mu(f)$ and hence Algorithm 1 correctly returns f'. □

Since each invocation of Algorithm 1 requires $O(n)$ time, Lemma 2 together with the previous discussions allows us to state the main theorem of this section:

Theorem 1. *There exists an algorithm that solves the ABSE-MS problem in $O(mn + n^2 \log n)$ time.*

4 An Algorithm for ABSE-AS

In this section we show how the ABSE-AS problem can be solved efficiently in $O(mn \log \alpha(m,n))$ time. Our approach first of all, in a preprocessing phase, computes in $O(m\, n \log \alpha(m,n))$ time all the replacement shortest paths from the source after the failure of every edge of T [15]. Then, the algorithm will run through $n-1$ phases, each returning in $O(m)$ time a BSE for a failing edge of T, as described in the following. Thus, the overall time complexity will be dominated by the preprocessing step.

Let us fix $e = (a,b) \in E(T)$ as the failing edge of T. The idea is to show that, after a $O(n)$ preprocessing time, we can compute the avg-stretch $\lambda(f)$ of any f in constant time. This immediately implies that we can compute a BSE for e by looking at all $O(m)$ swap edges for e.

Let $U = V(T_b)$ and let y be a node in U, we define:

$$M(y) = \sum_{v \in U} \frac{d_T(y,v)}{d_{G-e}(s,v)} \quad \text{and} \quad Q = \sum_{v \in U} \frac{1}{d_{G-e}(s,v)}.$$

Let $f = (x,y)$ be a candidate swap edge incident in $y \in U$. The avg-stretch of f can be rewritten as:

$$\lambda(f) = \sum_{v \in U} \frac{d_T(s,x) + w(f) + d_T(y,v)}{d_{G-e}(s,v)} = \big(d_T(s,x) + w(f)\big)Q + M(y).$$

Hence, the avg-stretch of f can be computed in $O(1)$ time, once Q and $M(y)$ are available in constant time. Observe that Q does not depend on y and can be computed in $O(n)$ time. The rest of this section is devoted to show how to compute $M(y)$ for every $y \in U$ in $O(n)$ overall time.

Computing $M(y)$ for all $y \in U$. Let y e y' be two nodes in U such that y is a child of y' in T. Moreover, let $U_y = V(T_y)$, and let $Q_y = \sum_{v \in U_y} \frac{1}{d_{G-e}(s,y)}$. Hence, we can rewrite $M(y)$ and $M(y')$ as follows:

$$M(y) = \sum_{v \in U_y} \frac{d_T(y,v)}{d_{G-e}(s,v)} + \sum_{v \in U - U_y} \frac{w(y,y') + d_T(y',v)}{d_{G-e}(s,v)}$$

and

$$M(y') = \sum_{v \in U_y} \frac{w(y,y') + d_T(y,v)}{d_{G-e}(s,v)} + \sum_{v \in U - U_y} \frac{d_T(y',v)}{d_{G-e}(s,v)}.$$

Therefore, we have:

$$M(y) = M(y') + w(y,y')\big(-Q_y + (Q - Q_y)\big) = M(y') + w(y,y')\big(Q - 2Q_y\big). \quad (1)$$

The above equation implies that $M(y)$ can be computed in $O(1)$ time, once we have computed $M(y')$, Q and Q_y. As a consequence, we can compute all

the $M(y)$'s as follows. First, we compute Q_y for every $y \in D$ in $O(n)$ overall time by means of a postorder visit of T_b. Notice also that $Q = Q_b$. Then, we compute $M(b)$ explicitly in $O(n)$ time. Finally, we compute all the other $M(y)$'s by performing a preorder visit of T_b. When we visit a node y, we compute $M(y)$ in constant time using (1). Thus, the visit will take $O(n)$ time. We have proved the following:

Theorem 2. *There exists an algorithm that solves the* `ABSE-AS` *problem in* $O(mn \log \alpha(m,n))$ *time.*

5 An approximate solution for ABSE-MS

In this section we show that for the max-stretch measure we can compute in an almost linear $O(m \log \alpha(m,n))$ time, a set of *good* swap edges (GSE), each of which guarantees a relative approximation factor on the maximum stretch of 3/2 (tight), as opposed to that provided by the corresponding BSE. Moreover, as shown in the next section, each GSE still guarantees an absolute maximum stretch factor w.r.t. the paths emanating from the source (in the surviving graph) equal to 3 (tight).

Lemma 3. *Let e be a failing edge in T, let*

$$g = (x,y) \in \arg \min_{(x',v') \in C(e)} \left\{ d_T(s,x') + w(x',v') \right\},$$

and, finally, let $f = (x',y')$ be a best swap edge for e w.r.t. `ABSE-MS`*. Then, $\mu(g)/\mu(f) \leq 3/2$.*

Proof. Let z be the critical node for the good swap edge g, and let t (resp., t') denote the *least common ancestor* in T between y' and z (resp., y' and y). Let $D = d_T(s,x) + w(x,y) = d_{G-e}(s,y)$. By choice of g, it holds that $d_{G-e}(s,z) \geq D$ and $d_{G-e}(s,y') \geq D$. We divide the proof into the following two cases, as depicted in Figure 2: either (1) t is an ancestor of t' in T, or (2) t' is an ancestor of t in T. Let A, B, C denote the distance in T between y and t', t' and t, t and z, respectively.

Case 1 Since t is an ancestor of t' (left side of Figure 2), we have that $d_{T_{e/f}}(s,y) \geq D + A$ and we can write:

$$\sigma_{G-e}(T_{e/f}, y) \geq \frac{D+A}{d_{G-e}(s,y)} = \frac{D+A}{D} \geq \frac{D+A}{d_{G-e}(s,z)},$$

and similarly $\sigma_{G-e}(T_{e/f}, z) \geq \frac{D+B+C}{d_{G-e}(s,z)}$. Moreover, by the definition of $\mu(\cdot)$ we have that $\mu(f) \geq \max\{\sigma_{G-e}(T_{e/f}, y), \sigma_{G-e}(T_{e/f}, z)\}$. The previous inequalities together imply:

$$\frac{\mu(g)}{\mu(f)} \leq \frac{\sigma_{G-e}(T_{e/g}, z)}{\max\{\sigma_{G-e}(T_{e/f}, y), \sigma_{G-e}(T_{e/f}, z)\}} \leq \frac{A+B+C+D}{D + \max\{A, B+C\}}. \qquad (2)$$

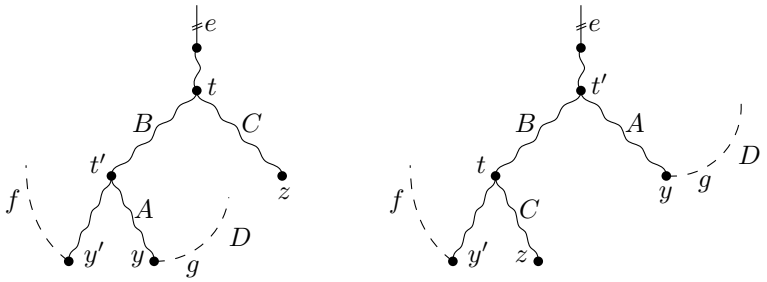

Fig. 2. The figure shows the two cases of the analysis, on the left t is an ancestor of t', while on the right the opposite holds. The splines denote a path, while the straight lines represent a single edge.

Now we divide the proof into two subcases, depending on whether $B+C \geq A$ or $B+C < A$. Observe that $D \geq d_G(s,y) \geq A$. If $B+C \geq A$, then (2) becomes:

$$\frac{\mu(g)}{\mu(f)} \leq \frac{A+B+C+D}{B+C+D} = 1 + \frac{A}{B+C+D} \leq 1 + \frac{A}{2A} = \frac{3}{2},$$

otherwise, if $B+C < A$, then (2) becomes:

$$\frac{\mu(g)}{\mu(f)} \leq \frac{A+B+C+D}{A+D} < \frac{2A+D}{A+D} = 1 + \frac{A}{A+D} \leq 1 + \frac{A}{2A} = \frac{3}{2}.$$

Case 2 Assume now that t' is an ancestor of t (right side of Figure 2). Since

$$\mu(f) \geq \sigma_{G-e}(T_{e/f}, y) \geq \frac{d_{G-e}(s,y') + A + B}{d_{G-e}(s,y)} = \frac{d_{G-e}(s,y') + A + B}{D},$$

we have that:

$$\begin{aligned}\frac{\mu(g)}{\mu(f)} &\leq \frac{A+B+C+D}{d_{G-e}(s,z)} \cdot \frac{D}{d_{G-e}(s,y') + A + B} \\ &\leq \frac{A+B+C+D}{d_{G-e}(s,z)} \cdot \frac{D}{A+B+D}\end{aligned}$$

and since $d_{G-e}(s,z) \geq d_G(s,z) \geq C$, and recalling that $d_{G-e}(s,z) \geq D$, we have:

$$\frac{\mu(g)}{\mu(f)} \leq \frac{A+B+C+D}{A+B+D} \cdot \frac{D}{\max\{C,D\}} = \left(1 + \frac{C}{A+B+D}\right) \cdot \frac{D}{\max\{C,D\}}. \quad (3)$$

Moreover, notice that also the following holds:

$$\begin{aligned}\frac{\mu(g)}{\mu(f)} &\leq \frac{\mu(g)}{\sigma_{G-e}(T_{e/f}, z)} \leq \frac{A+B+C+D}{d_{G-e}(s,z)} \cdot \frac{d_{G-e}(s,z)}{d_{G-e}(s,y') + d_T(y',t) + C} \\ &\leq \frac{A+B+C+D}{C+D} = 1 + \frac{A+B}{C+D}. \quad (4)\end{aligned}$$

We divide the proof into the following two subcases, depending on whether $D \geq C$ or $D < C$. In the first subcase, i.e., $D \geq C$, we have that (3) becomes $\frac{\mu(g)}{\mu(f)} \leq 1 + \frac{C}{A+B+D}$, and hence, by combining this inequality with (4), we obtain:

$$\begin{aligned}\frac{\mu(g)}{\mu(f)} &\leq 1 + \min\left\{\frac{C}{A+B+D}, \frac{A+B}{C+D}\right\} \\ &\leq 1 + \min\left\{\frac{C}{A+B+C}, \frac{A+B}{2C}\right\} \leq 1 + \frac{1}{2} = \frac{3}{2}.\end{aligned}$$

In the second subcase, i.e., $D < C$, (3) becomes:

$$\frac{\mu(g)}{\mu(f)} \leq \left(1 + \frac{C}{A+B+D}\right) \cdot \frac{D}{C} \leq \frac{D}{C} + \frac{D}{A+B+D} < 1 + \frac{D}{A+B+D}, \quad (5)$$

and hence, by combining (5) and (4), we have that:

$$\begin{aligned}\frac{\mu(g)}{\mu(f)} &\leq 1 + \min\left\{\frac{D}{A+B+D}, \frac{A+B}{C+D}\right\} \\ &\leq 1 + \min\left\{\frac{D}{A+B+D}, \frac{A+B}{2D}\right\} \leq 1 + \frac{1}{2} = \frac{3}{2},\end{aligned}$$

from which the claim follows. □

Given the result of Lemma 3, we can derive an efficient algorithm to compute all the GSE for `ABSE-MS`. More precisely, in [20] it was shown how to find them in $O(m\,\alpha(m,n))$ time. Essentially, the approach used in [20] was based on a reduction to the *SPT sensitivity analysis* problem [23]. However, in [22] it was proposed a faster solution to such a problem, running in $O(m \log \alpha(m,n))$ time. Thus, we can provide the following

Theorem 3. *There exists a* $3/2$*-approximation algorithm that solves the* `ABSE-MS` *problem in* $O(m \log \alpha(m,n))$ *time.*

We conclude this section with a tight example which shows that the analysis provided in Lemma 3 is tight (see Figure 3).

6 Quality analysis

As for previous studies on swap edges, it is interesting now to see how the tree obtained from swapping a failing edge $e = (a,b)$ with its BSE f compares with a true SPT of $G - e$. According to our swap criteria, we will then analyze the lower and upper bounds of the max- and avg-stretch of f, i.e., $\mu(f)$ and $\lambda(f)$, respectively.

As already observed in the introduction, it is well-known [20] that for the swap edge, say g, which belongs to the shortest path in $G-e$ between s and the root of the detached subtree T_b, we have that for any $v \in V(T_b)$, $\sigma_{G-e}(T_{e/g}, v) \leq 3$. This

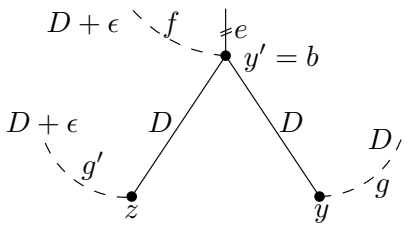

Fig. 3. A tight example showing that the quality of the good swap edge g computed by the algorithm is a factor of 3/2 away from the qualify of a best swap edge f. In the picture, it is assumed that the distance from s to b is equal to 0, while the three dashed edges are assumed to be incident to the source, and the labels correspond to their weight. Then, $\mu(f) = \sigma_{G-e}(T_{e/f}, y) = \frac{2D+\epsilon}{D} \simeq 2$, while $\mu(g) = \sigma_{G-e}(T_{e/g}, z) = \frac{3D}{D+\epsilon} \simeq 3$, for small values of ϵ.

immediately implies that $\mu(g), \lambda(g) \leq 3$, namely $\mu(f), \lambda(f) \leq 3$. These bounds happen to be tight, as shown in Figure 4.

Let us now analyze the lower and upper bounds of the max-stretch of a *good* swap edge g, i.e., $\mu(g)$, as defined in the previous section. First of all, once again it was proven in [19] that for any $v \in V(T_b)$, $\sigma_{G-e}(T_{e/g}, v) \leq 3$, which implies that $\mu(g) \leq 3$. Moreover, the example shown in Figure 4 can be used to verify that this bound is tight.

7 Conclusions

In this paper we have studied two natural SPT swap problems, aiming to minimize, after the failure of any edge of the given SPT, either the maximum or the average stretch factor induced by a swap edge. We have first proposed two efficient algorithms to solve both problems. Then, aiming to the design of faster algorithms, we developed for the maximum-stretch measure an almost linear algorithm guaranteeing a 3/2-approximation w.r.t. the optimum.

Concerning future research directions, the most important open problem remains that of finding a linear-size edge-fault-tolerant SPT with a stretch factor better than 3, or to prove that this is unfeasible. Another interesting open problem is that of improving the running time of our solutions. Notice that both our exact algorithms pass through the computation of all the post-failure single-source distances, and if we could avoid that we would get faster solutions. At a first glance, this sounds very hard, since the stretches are heavily dependant on post-failure distances, but, at least in principle, one could exploit some monotonicity property among swap edges that could allow to skip such a bottleneck. Finally, it would be nice to design a fast approximation algorithm for the average-stretch measure. Apparently, in this case it is not easy to adopt an approach based on good swap edges as for the maximum-stretch case, since swap edges optimizing other reasonable swap criteria (e.g., minimizing the distance towards the root of the detached subtree, or minimizing the distance towards a detached node)

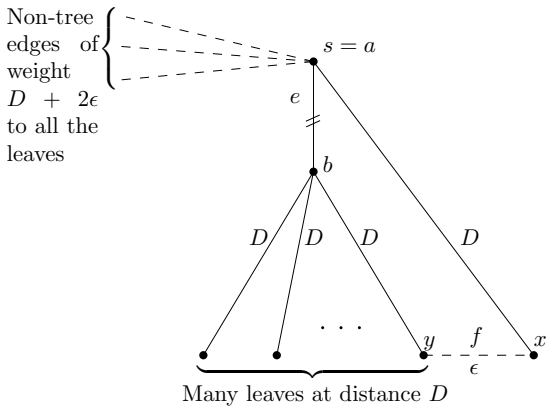

Fig. 4. Tight ratios for $\mu(f)$ and $\lambda(f)$. In the picture, the SPT T (solid edges) along with the removed edge $e = (a, b)$ of weight 0; non-tree edges are dashed and the best swap edge (for both the maximum and the average stretch) is easily seen to be $f = (x, y)$, of weight $\epsilon > 0$. Then, we have that $\mu(f) = \frac{3D+\epsilon}{D+2\epsilon}$, which tends to 3 for small values of ϵ, while $\lambda(f)$ tends to 3 as well, as soon as the number of leaves grows. Notice that f is also a good swap edge for e.

are easily seen to produce an approximation ratio of 3 as opposed to a BSE. A candidate solution may be that of selecting a BSE w.r.t. the sum-of-distances criterium, which can be solved in almost linear time [10], but for which we were unable to provide a corresponding comparative analysis.

References

1. Davide Bilò, Feliciano Colella, Luciano Gualà, Stefano Leucci, and Guido Proietti. A faster computation of all the best swap edges of a tree spanner. In *Structural Information and Communication Complexity - 22nd International Colloquium, SIROCCO 2015, Montserrat, Spain, July 14-16, 2015*, pages 239–253, 2015.
2. Davide Bilò, Fabrizio Grandoni, Luciano Gualà, Stefano Leucci, and Guido Proietti. Improved purely additive fault-tolerant spanners. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015*, pages 167–178, 2015.
3. Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-tolerant approximate shortest-path trees. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014*, pages 137–148, 2014.
4. Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Compact and fast sensitivity oracles for single-source distances. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 13:1–13:14, 2016.
5. Davide Bilò, Luciano Gualà, and Guido Proietti. Finding best swap edges minimizing the routing cost of a spanning tree. *Algorithmica*, 68(2):337–357, 2014.
6. Davide Bilò, Luciano Gualà, and Guido Proietti. A faster computation of all the best swap edges of a shortest paths tree. *Algorithmica*, 73(3):547–570, 2015.

7. Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault-tolerant spanners for general graphs. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 435–444, 2009.
8. Ajoy K. Datta, Lawrence L. Larmore, Linda Pagli, and Giuseppe Prencipe. Linear time distributed swap edge algorithms. In *Algorithms and Complexity, 8th International Conference, CIAC 2013, Barcelona, Spain, May 22-24, 2013*, pages 122–133, 2013.
9. Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.
10. Aleksej Di Salvo and Guido Proietti. Swapping a failing edge of a shortest paths tree by minimizing the average stretch factor. *Theor. Comput. Sci.*, 383(1):23–33, 2007.
11. Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 169–178, 2011.
12. Paola Flocchini, Antonio Mesa Enriques, Linda Pagli, Giuseppe Prencipe, and Nicola Santoro. Efficient protocols for computing the optimal swap edges of a shortest path tree. In *3rd International Conference on Theoretical Computer Science (IFIP-TCS'04), 22-27 August 2004, Toulouse, France*, pages 153–166, 2004.
13. Paola Flocchini, Antonio Mesa Enriques, Linda Pagli, Giuseppe Prencipe, and Nicola Santoro. Point-of-failure shortest-path rerouting: Computing the optimal swap edges distributively. *IEICE Transactions*, 89-D(2):700–708, 2006.
14. Paola Flocchini, Linda Pagli, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Computing all the best swap edges distributively. *J. Parallel Distrib. Comput.*, 68(7):976–983, 2008.
15. Luciano Gualà and Guido Proietti. Exact and approximate truthful mechanisms for the shortest paths tree problem. *Algorithmica*, 49(3):171–191, 2007.
16. Giuseppe F. Italiano and Rajiv Ramaswami. Maintaining spanning trees of small diameter. *Algorithmica*, 22(3):275–304, 1998.
17. Hiro Ito, Kazuo Iwama, Yasuo Okabe, and Takuya Yoshihiro. Single backup table schemes for shortest-path routing. *Theor. Comput. Sci.*, 333(3):347–353, 2005.
18. Camille Jordan. Sur les assemblages de lignes. *J. Reine Angew. Math*, 70(185):81, 1869.
19. Enrico Nardelli, Guido Proietti, and Peter Widmayer. A faster computation of the most vital edge of a shortest path. *Inf. Process. Lett.*, 79(2):81–85, 2001.
20. Enrico Nardelli, Guido Proietti, and Peter Widmayer. Swapping a failing edge of a single source shortest paths tree is good and fast. *Algorithmica*, 35(1):56–74, 2003.
21. Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013*, pages 779–790, 2013.
22. Seth Pettie. Sensitivity analysis of minimum spanning trees in sub-inverse-Ackermann time. In *Algorithms and Computation, 16th International Symposium, ISAAC 2005, Sanya, Hainan, China, December 19-21, 2005*, pages 964–973, 2005.
23. Robert Endre Tarjan. Sensitivity analysis of minimum spanning trees and shortest path trees. *Inf. Process. Lett.*, 14(1):30–33, 1982.
24. Bang Ye Wu, Chih-Yuan Hsiao, and Kun-Mao Chao. The swap edges of a multiple-sources routing tree. *Algorithmica*, 50(3):299–311, 2008.