Algorithmica (2022) 84:37–59
https://doi.org/10.1007/s00453-021-00879-8

Multiple-Edge-Fault-Tolerant Approximate Shortest-Path Trees

Davide Bilò[1] · Luciano Gualà[2] · Stefano Leucci[3] · Guido Proietti[3,4]

Received: 31 October 2019 / Accepted: 4 October 2021 / Published online: 30 October 2021

Abstract

Let G be an n-node and m-edge positively real-weighted undirected graph. For any given integer $f \geq 1$, we study the problem of designing a sparse f-edge-fault-tolerant (f-EFT) σ-approximate single-source shortest-path tree (σ-ASPT), namely a subgraph of G having as few edges as possible and which, following the failure of a set F of at most f edges in G, contains paths from a fixed source that are stretched by a factor of at most σ. To this respect, we provide an algorithm that efficiently computes an f-EFT $(2|F|+1)$-ASPT of size $O(fn)$. Our structure improves on a previous related construction designed for *unweighted* graphs, having the same size but guaranteeing a larger stretch factor of $3(f+1)$, plus an additive term of $(f+1)\log n$. Then, we show how to convert our structure into an efficient f-EFT single-source distance oracle, that can be built in $O(fm\,\alpha(m,n) + fn\log^3 n)$ time, has size $O(fn\log^2 n)$, and in $O(|F|^2 \log^2 n)$ time is able to report a $(2|F|+1)$-approximate distance from the source to any node in $G - F$. Moreover, our oracle can return a corresponding approximate path in the same amount of time plus the path's size. The oracle is obtained by tackling another fundamental problem, namely that of updating a minimum spanning forest (MSF) of G following a *batch* of k simultaneous modification (i.e., edge insertions, deletions and weight changes). For this problem, we build in $O(m\log^3 n)$ time an oracle of size $O(m\log^2 n)$, that reports in $O(k^2\log^2 n)$ time the (at most $2k$) edges either exiting from or entering into the MSF. Finally, for any integer $k \geq 1$, we complement all our results with a lower bound of $\Omega\left(n^{1+\frac{1}{k}}\right)$ to the size of any f-EFT σ-ASPT with $f \geq \log n$ and $\sigma < \frac{3k+1}{k+1}$, that holds if the Erdős' girth conjecture is true.

This work was partially supported by the Research Grant PRIN 2010 "ARS TechnoMedia", funded by the Italian Ministry of Education, University, and Research, and by the Research Grant "Algorithms for (fault-tolerant) pairwise spanners and distance oracles", funded by "Fondazione di Sardegna" in 2016. This work was partially supported by the "Fondo di Ateneo per la Ricerca 2019" Grant of the University of Sassari and by the project E89C20000620005 "ALgorithmic aspects of BLOckchain TECHnology" (ALBLOTECH).
A preliminary version of this work appeared in the proceeding of the 33rd International Symposium on Theoretical Aspects of Computer Science (STACS 2016).

Extended author information available on the last page of the article

Keywords Multiple-edge fault-tolerant shortest-path tree · Distance sensitivity oracle · Fully-dynamic minimum spanning tree

Mathematics Subject Classification 68P05 · 68W40

1 Introduction

Let $G = (V(G), E(G), w)$ be a positively real-weighted undirected graph of n nodes and m edges. A *shortest-path tree* (SPT) of G rooted at a distinguished source vertex, say s, is one of the most popular structures in communication networks. For example, it can be used for implementing the fundamental *broadcasting* operation. However, the SPT, as any tree-based topology, is highly sensitive to edge/vertex failures, which cause the undesired effect of disconnecting sets of vertices from the source.

Therefore, a general approach to cope with this scenario is to make the SPT resistant against a given number of component failures, by adding to it a set of suitably selected edges from the underlying graph, so that the resulting subgraph (also referred as *structure* in the following) will still contain a SPT of the surviving network. If we prepare ourselves to resist against a set of at most f failing edges in G, then the corresponding structure will be named an *f-edge-fault-tolerant* (f-EFT) SPT. Unfortunately, even if $f = 1$, $\Theta(m)$ additional edges may be needed, also in the case in which $m = \Theta(n^2)$ [12]. Thus, to sparsify such a structure, it makes sense to resort to *σ-approximate* shortest paths from the source, i.e., paths that are stretched at most by a factor $\sigma > 1$, for any possible set of failures that has to be handled (see Fig. 1 for an example).

In this paper, we show how to build[1] an efficient structure of this sort. Moreover, we show that it is possible to transform such a structure into an efficient *oracle* that will allow to quickly switch to the corresponding approximate replacement paths (or just to report their length).

1.1 Related Work

In the recent past, several single and multiple edge/vertex-fault-tolerant *approximate* SPT (ASPT) structures that offer different trade-offs between the guaranteed stretch and the overall size of the structure have been devised (see [39] for a survey). More formally, we say that a spanning subgraph H of G is an f-EFT σ-ASPT if it satisfies the following condition: For each set of edges $F \subseteq E(G)$ of size at most f, all the distances from the source s in the subgraph $H - F = (V(G), E(H) \setminus F, w)$ are at most σ times longer than the corresponding distances in $G - F$. If a further additive distortion β is also allowed to the distances, then the structure will be named f-EFT (σ, β)-ASPT. Similar definitions can be given for the *vertex-fault-tolerant* (VFT) case and for unweighted graphs (in this case we use the acronym ABFS instead of ASPT to stress the fact that we are dealing with unweighted graphs in which SPTs coincide with breadth-first search trees).

[1] Throughout this introduction, all the discussed structures are poly-time computable, even if we may omit to specify the actual running time.

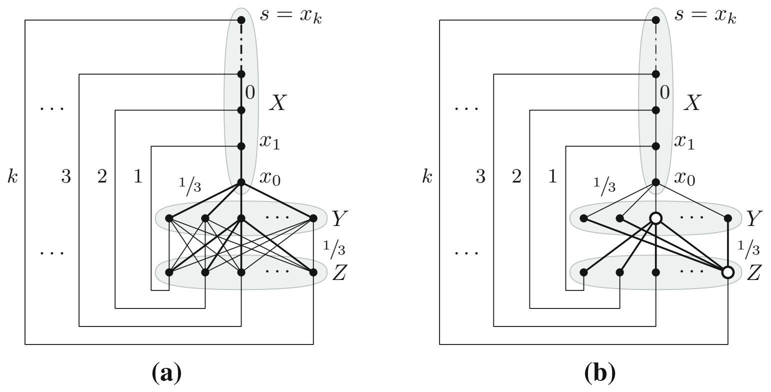


Fig. 1 **a** A graph G on n vertices and $\Theta(n^2)$ edges. The edges between vertices in X weight 0, while those incident to the vertices in Y have weight $\frac{1}{3}$. Each of the sets Y and Z contains $\Theta(n)$ vertices and $Y \cup Z$ induces a complete bipartite subgraph. The edges of an SPT of G from s are shown in bold. The only 1-EFT SPT H of G is G itself since, for $i = 1, \ldots, k$, an SPT of $G - (x_i, x_{i-1})$ contains the unique edge (x_i, z) of weight i together with all other edges incident to z. **b** A 1-EFT $\frac{3}{2}$-approximate SPT. The bipartite subgraph can be sparsified by only selecting the edges incident to the two white vertices (one from Y and one from Z). In general, if we want a f-EFT $\frac{3}{2}$-approximate SPT, we can still sparsify the bipartite subgraph by selecting all the edges incident to f vertices from Z and f vertices from Y

A natural counterpart of fault-tolerant SPT structures are fault-tolerant σ-*stretched single-source distance oracles* (σ-SSDO in the following), i.e., *compact* data structures that can be built with a *low* preprocessing time, and that are able to *quickly* return σ-approximate distances/paths from the source following a set of failures. Converting a fault-tolerant SPT into a corresponding SSDO with the very same stretch, and additionally having a small size and a fast query time, is a quite natural process, because of its practical usage: computing the alternative post-failure distances/paths on the structure may indeed be very time consuming. However, such a conversion process is not straightforward, in general, since it requires to exploit distance-related information that are instead implicit in the underlying structure, and this has to be done by optimizing the trade-off between the size and the query time of the oracle.

Turning back our attention to fault-tolerant SPT structures, their study originated in [6], where the authors showed the existence of a 1-VFT $(1+\varepsilon)$-ABFS with size $O(\frac{n}{\varepsilon^3} + n \log n)$, for any $\varepsilon > 0$, and of a 1-VFT 3-ASPT having size $O(n \log n)$. These results were obtained as byproducts of a 1-VFT 3-SSDO and a 1-VFT $(1+\varepsilon)$-SSDO of the same size, both having a distance (resp., path) query time of $O(1)$ (resp., proportional to the path's size). Later, in [12], the authors showed the existence of a 1-E/VFT $(1+\varepsilon)$-ASPT of size $O(\frac{n \log n}{\varepsilon^2})$, for any $\varepsilon > 0$ (without providing a corresponding oracle). In [11], the authors designed a $(1+\varepsilon)$-SSDO of size $O(n\varepsilon^{-1} \log \varepsilon^{-1})$ and query time $O(\log n)$, for any $\varepsilon > 0$. For the special case $\varepsilon = 2$, they also provided a simpler construction improving the query time to $O(1)$.

Concerning *unweighted* graphs, Parter and Peleg [41] presented a 1-E/VFT *Breadth-First Search tree* (BFS) of size $O(n \cdot \min\{ecc(s), \sqrt{n}\})$, where $ecc(s)$ denotes the eccentricity of the source vertex s in G, namely a structure containing *exact* shortest paths from the source after a single edge/vertex failure. In the same paper, the authors also exhibit a corresponding lower bound of $\Omega(n^{3/2})$ for the size of a 1-E/VFT BFS. Then, in [42], the same authors presented a set of lower and upper bounds to the size of (σ, β)-ABFS. More precisely, they showed that for every $\beta \in [1, O(\log n)]$, there exists a graph G and a source vertex $s \in V(G)$ such that a corresponding 1-EFT $(1, \beta)$-ABFS requires $\Omega(n^{1+\varepsilon(\beta)})$ edges, for some function $\varepsilon(\beta) \in (0, 1)$. Moreover, they also constructed a 1-EFT $(1, 4)$-ABFS of size $O(n^{4/3})$. Finally, assuming at most $f = O(1)$ edge failures can take place, they showed the existence of (i) an f-EFT $(3(f+1), (f+1)\log n)$-ABFS of size $O(fn)$, and (ii) an f-EFT $(3f+4)$-ABFS of size $O(fn\log^{f+1} n)$. These structures will be exactly our touchstone in this paper, since they are the only ones concerned with multiple-edge-failure single-source shortest paths.

1.2 Our Results

In this paper, we present the following main results:

- An f-EFT $(2|F|+1)$-ASPT of size $O(fn)$ that is able to handle the failure of any set $F \subseteq E(G)$ of at most f edges. This considerably improves w.r.t. to its direct competitors, namely the structures presented in [42]: our structure has a size that is never worse, a lower stretch, works on weighted graphs, and handles an arbitrary (i.e., even non-constant) number of failures. Moreover, our construction is simpler and can be computed quickly in $O(fm\,\alpha(m, n))$ time, where α is the inverse of the Ackermann's function.
- A corresponding f-EFT $(2|F|+1)$-SSDO of size $O(fn\log^2 n)$, that can be built in $O(fm\,\alpha(m, n) + fn\log^3 n)$ time, has a distance query time of $O(|F|^2 \log^2 n)$, and is also able to report the corresponding path in the same time plus the path's size. Moreover, if one is willing to use $O(m\log^2 n)$ space, then our oracle can handle any number of edge failures (i.e., up to m).

Interestingly enough, the former result is obtained by posing a simple yet surprising relationship between the structure of the replacement paths and the *minimum spanning forest* (MSF) of an ad-hoc auxiliary graph. This approach is also useful to develop the latter result, that is indeed obtained through an efficient updating of an MSF after that a *batch* of any number k of edge modifications (i.e., edge insertions, deletions and weight changes) are simultaneously performed. For this problem indeed we provide the following result:

- a (multiple-update) *MSF sensitivity oracle*[2] of size $O(m\log^2 n)$, that can be built in $O(m\log^3 n)$ time, and is able to report in $O(k^2\log^2 n)$ time the (at most $2k$) edges either exiting from or entering into the MSF. As a result of independent interest, it is worth noticing that our oracle can be used to efficiently maintain a MSF under relatively short sequences of *non-simultaneous* updates. Indeed, observe that a sequence $\lambda = \langle \lambda_1, \ldots, \lambda_h \rangle$ of updates can be managed through h sequential queries to the oracle, where the i-th query will involve the modifications

[2] We use this terminology for the oracle in accordance with its functionality of only reporting the topological changes in the MSF.

to the starting MSF induced by the batch of the first i updates. This way, we spend $O(h^2 \log^2 n)$ time to handle each single update. Hence, as the fastest long-standing algorithm for the classic (and clearly more general) *fully-dynamic* MSF problem has a worst-case cost of $O(\sqrt{n})$ per update [25], it follows that for $h = o(\sqrt[4]{n}/\log n)$, our oracle should be preferred, since it will manage each update in $o(\sqrt{n})$ time. Notice also that a comparison with other known online/offline algorithms for maintaining an MSF that are more efficient in an amortized sense, like for instance those given in [24,33,34], is unfeasible. Indeed these algorithms need to start from an empty graph to guarantee their bounds or they need long sequences of updates to become efficient. Thus, when starting from an arbitrary graph, as it happens in our setting, a single update operation could even cost them $\Theta(n)$ time!

Finally, for any integer $k \geq 1$, we prove a lower bound of $\Omega(n^{1+\frac{1}{k}})$ on the size of any f-EFT σ-ASPT with $f \geq \log n$ and $\sigma < \frac{3k+1}{k+1}$, that holds if the Erdős' girth conjecture is true. Our lower bound shows that, in contrast to the single-edge failure case, it is not possible to obtain a stretch arbitrary close to 1 with size $\widetilde{O}(n)$[3] when the number of faults is more than $\log n$. We look at the problem of understanding whether this can be done for constant $f > 1$ as an interesting open problem.

1.3 Other Related Work on Fault-Tolerant Sourced Structures/Oracles

A vast body of literature deals with structures and oracles that tolerate single/multiple failures in single-source shortest paths. An early work on the topic is [37], where the authors were concerned with the computation of *best swap edges* (w.r.t. several swap functions) for the failure of each and every edge in a SPT. As a by-product of their results, it can be easily seen that by adding to a SPT the (at most) $n - 1$ best swap edges w.r.t. to the new distance from s to the root of the subtree disconnected from s after an edge failure, then a 1-EFT 3-ASPT is obtained. Interestingly, such a structure can be easily converted into a 1-EFT 3-SSDO of size $O(n)$ and query time $O(1)$. In [21], the authors faced the special case of *shortest-path failures*, in which the failure of a set F of at most f adjacent edges along any source-leaf path has to be tolerated. They proposed an f-EFT $(2k-1)(2|F|+1)$-ASPT of size $O(kn\, f^{1+1/k})$, where $|F|$ denotes the size of the actual failing path, and $k \geq 1$ is a parameter of choice. Notice that this result is subsumed by ours. Moreover, they also provided a conversion to a corresponding oracle, and for the special case of $f = 2$, they gave an ad-hoc solution of size $O(n \log n)$ and with stretch 3. For directed graphs with integer positive edge weights bounded by M, in [28] the authors showed how to build efficiently in $\widetilde{O}(Mn^{\omega})$ time a randomized 1-EFT 1-SSDO of size $\Theta(n^2)$ and with $O(1)$ query time, where returned distances are exact w.h.p., and $\omega < 2.373$ denotes the matrix multiplication exponent.

Concerning unweighted graphs, in [12] the authors showed that an *ordinary* (i.e., non fault-tolerant) (σ, β)-*spanner* (i.e., where distances/paths between arbitrary pairs of nodes are at most (σ, β)-stretched) of size $O(g(n))$ can be used to build a 1-EFT (resp., VFT) (σ, β)-ABFS of the same size (resp., of size $O(g(n) + n \log n)$). This

[3] The $\widetilde{O}$ notation hides poly-logarithmic factors in n.

result is useful for building sparse 1-VFT $(1, \beta)$-ABFS structures by making use of the vast literature on *additive* $(1, \beta)$-spanners (e.g., [7,18]). In [38], Parter presented a 2-EFT BFS having $O(n^{5/3})$ edges, which is tight.

Another research stream related to our work is that on *multi-source* fault-tolerant structures,[4] for which we look at distances/paths from a set $S \subseteq V(G)$ of sources. Here, results are known only for unweighted graphs. For $f = 1$, [41] shows how to compute a 1-EFT MSBFS of size $O(\sqrt{|S|}\, n^{3/2})$, which is tight and has been converted, among other results, to an oracle having size $\widetilde{O}(\sqrt{|S|}\, n^{3/2})$ that is able to report post-failure shortest-paths in $O(1)$ time per edge [9]. In [30], Gupta and Singh showed that post-failure distance queries can be answered in time $O(\text{polylog} n)$ without increasing the oracle's size. For $f = 2$ the construction of [38] can be generalized to yield a 2-E/VFT MSBFS of size $O(|S|^{\frac{1}{3}} n^{\frac{5}{3}})$, also for the case of directed graphs [29]. For large values of f, [13] shows how to build a f-E/VFT MSBFS of size $\widetilde{O}(f \cdot |S|^{1/2^f} n^{2-1/2^f})$, which quickly approaches $\Theta(n^2)$ even in the single-source case. As the authors show, this is unavoidable: for any fixed $\varepsilon > 0$, there is a large enough value f (depending on ε) for which all f-E/VFT BFS constructions must use $\Omega(n^{2-\varepsilon})$ edges in the worst case, and this is true even if a constant additive error is allowed. In [12], it was shown that an ordinary (σ, β)-spanner of size $O(g(n))$ can be used to build a 1-EFT (σ, β)-AMSBFS of size $O(g(n) + |S|\, n)$, and similarly for the vertex case of size $O(g(n) + |S|\, n \log n)$.

1.4 More Related Work on (Fault-Tolerant) Spanners/Oracles

For the sake of completeness, we also give some hints on the large body of literature on the related topic of (fault-tolerant) spanners and distance oracles.

As far as spanners are concerned, on weighted graphs the current best known construction is, for any $f \geq 1$ and any integer parameter $k \geq 1$, the f-E/VFT $(2k-1)$-spanner of size $O(f^{1-\frac{1}{k}} n^{1+\frac{1}{k}})$ given in [14]. For a comparison, the sparsest known $(2k-1)$-multiplicative ordinary spanner has size $O(n^{1+1/k})$ [3], and this is believed to be asymptotically tight due to the girth conjecture of Erdős [26]. In [4] the authors introduced the related concept of 1-EFT *resilient* spanners, i.e., spanners that approximately preserve the multiplicative increment in distances following an edge failure. In unweighted graphs, fault-tolerant additive spanners were also considered. In particular, Braunshvig et al. [15] proposed the following general approach to build an f-EFT additive spanner: Let A be an f-EFT σ-spanner, and let B be an ordinary $(1, \beta)$-spanner. Then $H = A \cup B$ is an f-EFT $(1, 2f(2\beta + \sigma - 1) + \beta)$-spanner. The corresponding analysis has been refined in [10] yielding a better additive bound of $2f(\beta + \sigma - 1) + \beta$.

As far as distance oracles are concerned, ordinary (i.e., fault-free) *all-pairs distance oracles* (APDO) on weigthed graphs were introduced in a seminal work by Thorup and Zwick [44] (who also coined the term *oracle*), followed by a sequel of papers (among the others, we mention [19,23] for the currently best bounds). In a fault-tolerant setting, in [8] the authors built (on directed graphs) a 1-E/VFT 1-APDO of size $\widetilde{O}(n^2)$ and with query time $O(1)$. For two failures, in [22] the authors built, still

[4] These structures are also known as *sourcewise spanners*. We will write MSBFS and MSABFS in place of *multi-source BFS* and *multi-source ABFS*, respectively.

on directed graphs, a 2-E/VFT 1-APDO of size $\widetilde{O}(n^2)$ and with query time $O(\log n)$. Concerning multiple-edge failures, in [20] the authors built, for any integer $k \geq 1$, an f-EFT $(8k-2)(f+1)$-APDO of size $O(fkn^{1+1/k}\log(nW))$, where W is the ratio of the maximum to the minimum edge weight in G. The query time of the oracle is $\widetilde{O}(|F| \log\log d)$, where F is the actual set of failing edges, and d is the distance between the queried pair of nodes in $G - F$.

Finally, for other results on single edge/vertex failures spanners/oracles, we refer the reader to [6,9,10,16,40] and to [1] for a survey.

1.5 Structure of the Paper

The paper is organized as follows. First of all, in Sect. 2 we present the MSF sensitivity oracle, that will be instrumental to obtain our efficient oracle. Then, in Sect. 3, we introduce the algorithm for building our structure, and the associated oracle, while in Sect. 4 we present the claimed lower bound. Finally, in Sect. 5 we provide some concluding remarks and hints for possible future works.

2 A Minimum Spanning Tree *k*-Sensitivity Oracle

In this section we present an oracle that, given a real-weighted graph G, along with any *minimum spanning tree* (MST) T of G,[5] is able to answer queries of the form:

> Given a set F of k edge updates on G (i.e., edge insertions, deletions and weight modifications), let T' be a new MST of G. What are the edges in the symmetric difference of $E(T)$ and $E(T')$?

In other words, the oracle can report all the edges of T that leave the MST as a consequence of the updates, along with all the new edges in T' that enter the MST in their place. The oracle has a size of $O(m \log^2 n)$ and can be built in $O(m \log^2 n)$ space and $O(m \log^3 n)$ time, while a query involving k updates can be answered in $O(k^2 \log^2 n)$ time and space.

Our oracle exploits the fact that, when few updates are to be handled, the changes in the resulting MST will be small. This implies that large portions of T and T' will coincide, hence finding and reusing these portions allows us to save a considerable amount of work compared to the time needed to recompute T' from scratch. To this aim, we build a structure that maintains a set of connected subtrees of T at different levels of granularity.

A high-level description of our approach is the following. First of all, notice that since a weight modification can be simulated by a deletion followed by an insertion, and since insertions can be managed in $O(\log n)$ time using known data structures

[5] For the sake of avoiding technicalities, in the following we assume that each edge is subject to at most a single update and we also assume that the graph G always remains connected, so that we simply talk about a MST instead of a MSF of G. For instance, this can be easily guaranteed by adding a dummy vertex $x \notin V(G)$ that is connected to all the vertices of $V(G)$ with edges of large weights.

(details can be found in 2.3), it turns out that the difficulty of the problem lies in handling deletions. For them, we perform the following two preprocessing steps:

(i) We compute a *hierarchical clustering* of the vertices of T;
(ii) We associate with each pair of clusters, say C, C', a set of *crossing edges*, namely edges of G having one endvertex in C and the other in C'.

Then, whenever a query involving a set F of edge updates is carried out on the oracle, we build an auxiliary graph $\widetilde{G}$ whose vertices correspond to the preprocessed clusters, and whose edges are suitably selected from the set of crossing edges – depending on which edges were removed in F. Finally, we compute a MST of $\widetilde{G}$, from which the query associated with F will be answered.

In the following section, we present the building blocks of our approach step by step.

2.1 Building the Clustering

Let Δ be the maximum degree of a vertex in the given MST T of G.[6] Our clustering $\mathcal{C}$ is a laminar family over $V(T)$, i.e., a collection of clusters (i.e., subsets of $V(T)$) such that any two clusters from $\mathcal{C}$ are either disjoint or one is contained in the other. Moreover, $\mathcal{C}$ will guarantee the following properties:

P_1. $V(T) \in \mathcal{C}$, $\emptyset \notin \mathcal{C}$ and, for each $v \in V(T)$, $\{v\} \in C$.
P_2. Each cluster $C \in \mathcal{C}$ with $|C| > 1$ can be partitioned into at least 2 and at most $\Delta + 1$ other clusters from $\mathcal{C}$.
P_3. If $C, C' \in \mathcal{C}$ and $C \subset C'$ then $2|C| \leq |C'|$.
P_4. The vertices in each cluster induce a connected component of T;

From P_3 and the fact that $\mathcal{C}$ is a laminar family it follows that the number of clusters $C \in C$ containing any single vertex v are at most $\log n$. Figure 2a shows an example of such a clustering $\mathcal{C}$. We can represent $\mathcal{C}$ with a tree $\mathcal{T}$ in which each vertex is a cluster. The tree $\mathcal{T}$ is rooted in the cluster $V(T)$, its leaves are the singleton clusters in $\mathcal{C}$, and the children of each internal node C are the maximal clusters $C' \in \mathcal{C}$ such that $C' \subset C$ (see Fig. 2b).[7] Notice that, since each internal vertex has at least two children (by P_2) and the number of leaves is n, we immediately have $|\mathcal{C}| = |V(\mathcal{T})| = O(n)$.

We now show describe a recursive algorithm that computes a clustering of T satisfying properties P_1–P_4. A generic invocation of our algorithm receives a subtree T' of T as its input and adds a clustering of T' to a *global* clustering $\mathcal{C}$ of T, which is initially empty. The algorithm begins by adding $V(T')$ to $\mathcal{C}$ and then checks whether T' consists of a single vertex. If this is the case we are done, otherwise we compute a centroid[8] v of T' and we invoke the algorithm recursively on each of the trees $T_1, T_2, \ldots, T_k$ of the forest obtained by deleting v from T. Our clustering $\mathcal{C}$ is the

[6] Since T is an unrooted tree, we define the degree of a vertex v in T to be the number of edges that are incident v.

[7] A cluster C' is maximal if there is no other cluster $C'' \in \mathcal{C}$ such that $C' \subset C'' \subset C$.

[8] A centroid v of a tree T' is a vertex v such that each tree in the forest $T' - v$ has at most $|V(T')|/2$ vertices. Each tree has either one or two centroids [35,36].

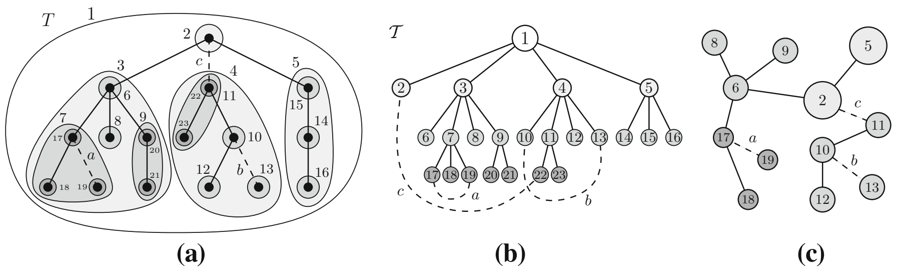


Fig. 2 **a** The MST T, along with the set of edges of T (dashed) which are deleted in F (for the sake of readability, the remaining edges of G are not depicted); the picture shows all the sets of $\mathcal{C}$, which are numbered. **b** the tree $\mathcal{T}$ associated with the hierarchical clustering $\mathcal{C}$. **c** the set of vertices of $\widetilde{G}$ computed by Algorithm 1. The dashed edges in **c** correspond to the deletions in F, so they do not belong to $\widetilde{G}$; solid lines represent edges of T that will appear in $\widetilde{G}$. The remaining edges of $\widetilde{G}$ are selected from the set of crossing edges and are not shown

result of the execution of the above algorithm with input T. Observe that the tree $\mathcal{T}$ associated with $\mathcal{C}$ is closely related to a centroid decomposition of T.

The following lemma bounds the time complexity of our recursive algorithm.

Lemma 1 *The above algorithm for computing $\mathcal{C}$ requires time $O(n \log n)$.*

Proof We start by noticing that the centroid of a tree can be found in linear time in the number of the tree's vertices. We let $c > 0$ be a large enough constant such that (i) the non-recursive part of the algorithm can be executed in time at most cn when the input tree has n vertices (the non-recursive part includes the computation of the centroid), and (ii) the overall time required by the algorithm when the input tree has at most 3 vertices is at most c.

We now prove by induction on $n \geq 4$ that the worst-case time $S(n)$ required by the algorithm when the input tree T has n vertices is at most $cn \log n$. Let v be the centroid of T, let $k \leq D$ be the degree of v in T, let $T_1, \ldots, T_k$ be the trees of the forest $T - v$, and denote by n_i the number of vertices in T_i. By the properties of the centroid, we know that each n_i is at most $\frac{n}{2}$. If $n_i \leq 3$ then $T(n_i) \leq c \leq cn_i(\log n - 1)$ by our choice of c, otherwise we can invoke the induction hypothesis to write $T(n_i) \leq cn_i \log n_i \leq cn_i(\log n - 1)$. Since the time spent by the algorithm is at most $cn + \sum_{i=1}^{k} T(n_i)$, we have: $cn + \sum_{i=1}^{k} T(n_i) \leq cn + c(\log n - 1) \sum_{i=1}^{k} n_i = cn + c(\log n - 1)(n - 1) < cn \log n$. □

2.2 Computing the Crossing Edges

In order to build the auxiliary graph that will be incorporated in our oracle, we need to map the edges of G onto the clustering described above, as explained in the following.

More in detail, we associate with each pair of clusters C, C' with $C \neq C'$ a list $E(C, C')$ containing all the edges of $E(G)$ with one endpoint in C and the other in C'. This list is ordered according to edge weights in a non-decreasing fashion. Let $C(u) = \{C \in \mathcal{C} : u \in C\}$ be the set of clusters that contain vertex u. As we already observed, we must have $|C(u)| = O(\log n)$, therefore each edge $(u, v) \in E(G)$ appears in at most $|C(u)| \cdot |C(v)| = O(\log^2 n)$ clusters showing that the overall number of elements in the lists is at most $O(m \log^2 n)$. We now show that these lists can be built in time $O(m \log^3 n)$ and stored in a way that allows the list associated with a given pair of clusters to be accessed in $O(1)$ time (which will guarantee the promised query time of our oracle).

Our construction maintains a dictionary D, whose keys will be unordered pair of clusters and whose values will be pointers to the corresponding lists of edges. Initially D is empty. We first sort all the edges of G in non-decreasing order of weight, prioritizing the edges in T whenever ties arise, and we examine one edge at a time. When $e = (u, v)$ is considered, we use the tree $\mathcal{T}$ to find all the clusters $C_u^1, C_u^2, \dots, C_u^h$ (resp. $C_v^1, C_v^2, \dots, C_v^\ell$) that contain u but not v (resp. v but not u). In order to do so, we compute the *lowest common ancestor* (LCA) x of the clusters $\{u\}$ and $\{v\}$ in $\mathcal{T}$. Then, $C_u^1, C_u^2, \dots$ (resp. $C_v^1, C_v^2, \dots$) are exactly the vertices in the unique path from $\{u\}$ (resp. $\{v\}$) to x (excluded) in $\mathcal{T}$. For each pair $\{C_u^i, C_v^j\}$, with $i = 1, \dots, h$ and $j = 1, \dots, \ell$, we query D: if the key $\{C_u^i, C_v^j\}$ exists, then we add e to the corresponding list, otherwise we create a new list $E(C_u^i, C_v^j)$ containing e and we add to D a new element with $\{C_u^i, C_v^j\}$ as its key, and a pointer to $E(C_u^i, C_v^j)$ as its value.

The above procedure requires $O(m \log^3 n)$ time, as each edge belongs to $O(\log^2 n)$ pairs of clusters, as observed before, and a query on D requires $O(\log n)$ time. However, for efficiency reasons, we will need for our oracle to have a constant access time to the list associated with a given pair of clusters. This can be obtained by building a *static* version of the dictionary D. This can be done in $O(\eta \log \eta)$ time by using the approach given in [31], where η is the number of elements in the original dictionary. In our case $\eta = O(m \log^2 n)$, hence the overall time needed to handle the crossing edges remains $O(m \log^3 n)$, using $O(m \log^2 n)$ space.

2.3 Answering a Query

We are now ready to describe how a query can be answered. In order to do so, it is useful to split each weight update operation involving an edge e into two separate operations, namely the deletion of e followed by its reinsertion with the new (updated) weight. By doing so, all the operations in F are now either insertions or deletions. For the sake of clarity, we first consider the case in which all the updates F are edge deletions, and we will show later how this can be extended to deal also with edge insertions. Moreover, we initially provide a solution whose size and query time depend on the degree of T, and then we show how this dependence can be avoided, yielding the claimed result.

Handling Edge Deletions In order to handle deletions, we use Algorithm 1 to construct an auxiliary graph $\widetilde{G}$ whose vertices are clusters. The algorithm computes a set R of clusters of $\mathcal{T}$ that coincides with $V(\widetilde{G})$. Initially R contains the unique cluster in $V(T) \in C$ that is the root of $\mathcal{T}$ and represents the whole tree T. At each time, the set of clusters in R will always form a partition of the vertices in V. The algorithm proceeds iteratively, by considering one after the other the edges of $F = \{e_1, \dots, e_k\}$. When an edge $e_i = (u, v)$ is considered, if $e_i \in E(T)$ and u and v belong to the same

Algorithm 1: Algorithm for computing the set of vertices (i.e., clusters) of $\widetilde{G}$.

```
1 $R \leftarrow \{V(T)\}$
2 for $(u, v) \in F$ do
3     if $(u, v) \in E(T)$ then
4         while $\text{root}_{\mathcal{T}}(\{u\}) = \text{root}_{\mathcal{T}}(\{v\})$ do
5             $C \leftarrow \text{root}_{\mathcal{T}}(\{u\})$
6             $R \leftarrow (R \setminus \{C\}) \cup \text{children}_{\mathcal{T}}(C)$        // Split C
7             $\mathcal{T} \leftarrow$ Delete $C$ from $\mathcal{T}$
8 return $R$
```

cluster C of R, we *split* C, i.e., we remove C from $\mathcal{T}$ and R, and we add to R all the maximal clusters of $\mathcal{C}$ that are strictly contained in C, i.e, the former children of C in $\mathcal{T}$. In this way $\mathcal{T}$ is always a forest and R contains the roots of the trees in $\mathcal{T}$. In the pseudocode of Algorithm 1 we use $\text{root}_{\mathcal{T}}(C')$ to denote the root of the tree in $\mathcal{T}$ that contains cluster C', and $\text{children}_{\mathcal{T}}(C')$ to denote the set of children of C' in $\mathcal{T}$.

In the end, $\widetilde{G}$ is such that all the edges in F have their endvertices into different clusters of $V(\widetilde{G})$. Moreover, by P_4, the clusters in $V(\widetilde{G})$ are associated with connected fragments of T. Since each edge in F can cause at most $O(\log n)$ splits (as this is also an upper bound on the height of $\mathcal{T}$), and each split operation can increase the number of vertices by at most Δ (by P_2 and the definition of $\mathcal{T}$), we have that $\widetilde{G}$ contains at most $O(|F|\Delta \log n) = O(k\Delta \log n)$ vertices (see Fig. 2c). Notice that, given a failing edge $(u, v) \in E(T)$, the clusters of $\mathcal{T}$ that need to be split are those in the unique path from the root of $\mathcal{T}$ to the lowest common ancestor of (the singleton clusters containing) u and v. Therefore such clusters can be found in $O(k \log n)$ time and an efficient implementation of Algorithm 2 requires $O(k\Delta \log n)$ time.

To construct the set $E(\widetilde{G})$ we consider all the pairs C, C' of vertices in $V(\widetilde{G})$. For each of these $O(\Delta^2 k^2 \log^2 n)$ pairs, we examine the edges in $E(C, C')$, in order, and we select the first edge e so that $e \notin F$, if any. Then, if e exists, we add the edge (C, C') to $\widetilde{G}$ with weight $w(e)$. Notice that e is an edge of minimum weight among those in $G - F$ that have one endvertex in C and the other in C'. Moreover, thanks to the static dictionary, all the above edges can be found in $O(k + \Delta^2 k^2 \log^2 n) = O(\Delta^2 k^2 \log^2 n)$ time (the additional additive term k accounts for the edges in the lists that belong to F).

We can now compute a MST $\widetilde{T}$ of $\widetilde{G}$ in time $O(\Delta^2 k^2 \log^2 n)$ by using any standard MST algorithm. We claim that once that the vertices (i.e., clusters) of $\widetilde{T}$ are expanded to their MST fragments, the resulting structure is exactly a MST of $G - F$. Indeed, by the cut property of MSTs, we know that all the edges of $T - F$ (which are a superset of the intra-cluster edges) also belong to an MST of $G - F$. It is well-known that contracting edges from an MST and computing an MST of the resulting graph yields an MST of the original graph [43, Ch. 11.5.2]. In our case we are contracting all intra-cluster edges and the resulting graph is exactly $\widetilde{G}$. Then, we look at the edges of $\widetilde{T}$ and we answer the query by returning the edges of G corresponding to those in $E(\widetilde{T})$ that are not in $E(T)$.

Notice that once the MST $\widetilde{T}$ of the auxiliary graph $\widetilde{G}$ has been computed, it can be used to answer the following additional query $q(u, v)$ in $O(\Delta^2 k^2 \log^2 n)$ time. These queries will be needed by our oracle of Sect. 3.1.

> Given a set F of at most f edge failures in G, and given a pair of nodes $u, v \in V(G)$, report all the new edges (and their weights) w.r.t. T on the unique path from u to v in a new MST of G, in order.

Indeed, these new edges can be detected by simply checking the path in $\widetilde{T}$ between the corresponding clusters containing u and v.

Handling General Edge Updates It turns out that the difficulty of the problem lies in handling the edge-deletion operations. Indeed, once this has been done, the remaining edge-insertion operations can be easily performed. To this aim, we reorganize the batch F (of size k) by first performing all the deletion operations, and we make use of a *top-tree* [2], i.e., a data structure that dynamically maintains a (weighted) forest under edge-insertion (*link*) and edge-deletion (*cut*) operations. Moreover, given two vertices u and v, top-trees are able to report the heaviest edge that lies on the path between u and v in the current forest. Each of these operations can be performed in $O(\log \eta)$ time where η is the number of vertices of the forest.

The idea is to maintain the current MST T' by using a top-tree that is initialized when the oracle is built to represent the tree T. This takes $O(n \log n)$ time. Then, we perform all the edge-deletion operations (as already described), while updating the top-tree accordingly (this requires only $O(k \log n)$ additional time since the number of needed link and cut operations is $O(k)$).

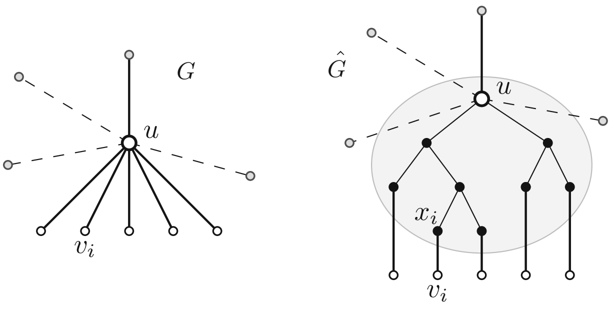
Now we handle the insertions one by one. In order to insert a new edge $e = (u, v)$, we search for the heaviest edge e' of the path connecting u and v in T'. If e' is heavier than e, we cut e' from T' and we link the two resulting components by adding the edge e. It is easy to see that this procedure requires an overall time of $O(k \log n)$.

By keeping track of all the $O(k)$ updates in the MST T', we can easily answer a query consisting of both edge-insertion and edge-deletion operations in $O(\Delta^2 k^2 \log^2 n)$ time, by reporting all the edges in the symmetric difference of T and T'.

Reducing the Degree of T So far, the complexity of our MST oracle depends on the maximum degree Δ of the vertices in T. However, using standard techniques (see, e.g., [27]), we now show that the updates on the original graph G and its MST T can be mapped onto an auxiliary graph $\hat{G}$ with weight function $\hat{w}$ and a corresponding MST $\hat{T}$, such that $\hat{G}$ has asymptotically the same size of G, and each vertex of $\hat{G}$ has a degree at most 3 in $\hat{T}$.

Initially $\hat{G}$, $\hat{w}$, and $\hat{T}$ coincide with G, w, and T, respectively. We root $\hat{T}$ in an arbitrary vertex, we iteratively search for a vertex u in $\hat{T}$ that has more than 2 children, and we lower its degree. We let $v_1, \ldots, v_h$ be the children of u in $\hat{T}$, and we proceed as follows: we remove all the edges in $\{(u, v_i) : 1 \leq i \leq h\}$ from both $\hat{G}$ and $\hat{T}$, then we add to both $\hat{G}$ and $\hat{T}$ a binary tree whose root coincides with u, and that has exactly h leaves $x_1, \ldots, x_h$. We assign weight $\hat{w}(e) = -\infty$ to all the edges e of this tree.[9] Finally, we add to $\hat{G}$ and $\hat{T}$ an edge (x_i, v_i) for each $1 \leq i \leq h$, we

[9] Here we use $+\infty$ (resp., $-\infty$) to denote a special edge weight that is always considered to be larger (resp., smaller) than any other edge weight.



(a) **(b)**

Fig. 3 Reducing the degree of vertices in T: on the left side, the tree T (solid edges) embedded in G, on the right side the superimposition of the binary tree to T in order to get a maximum degree of 3. Thin solid edges have weight 0, while the weight of (x_i, v_i) is $w(u.v_i)$

set $\hat{w}(x_i, v_i) = w(u, v_i)$, and we label (x_i, v_i) with $\mu((x_i, v_i)) = (u, v_i)$. For all the edges e of $\hat{G}$ not affected by this transformation (i.e., the ones that also belong to G) we set $\mu(e) = e$. An example of such a transformation is shown in Fig. 3. Let Z be the set of all the edges of weight $-\infty$ added to $\hat{G}$ by this procedure.

Each time we have to perform a weight update or delete operation on an edge (u, v_i) of G, we instead perform it on the corresponding edge $(x_i, v_i) = \mu^{-1}(u, v_i)$. Insertions and operations involving edges in $E(G) \setminus E(T)$ do not require any special care. In a similar way, whenever the answer of a query contains an edge (x_i, v_i), we replace it with the corresponding edge (u, v_i). Clearly, $O(n)$ vertices and edges are added by this process, and hence $|V(G)| = \Theta(|V(\hat{G})|)$ and $|E(G)| = \Theta(|E(\hat{G})|)$.

For the sake of completeness we now formally prove that a MST of $\hat{G}$ induces a MST of G even after performing a (possibly empty) set of update operations on both G and $\hat{G}$. To this aim it will be helpful to think of both G and $\hat{G}$ as complete graphs where the functions w and $\hat{w}$ assigning weight $+\infty$ to all the edges that are not in the original graphs. By doing so, applying a set of updates to G and $\hat{G}$ can be seen as a replacing the old weight functions with new ones, say w' and $\hat{w}'$, respectively. Notice also that, by doing so, for each edge $e \in Z$ we have $\hat{w}'(e) = \hat{w}(e) = -\infty$ while both $\hat{w}'(e) > -\infty$ and $\hat{w}(e) > -\infty$ if $e \notin Z$.

Fact 1 *Let M' be a MST of $\hat{G}$ w.r.t. $\hat{w}'$. The edges in $E(M') \setminus Z$ induce a MST of G w.r.t. w'.*

Proof In the rest of the proof, we assume that whenever ties arise they will be broken in a consistent way. Notice that the edges in Z form a forest in $\hat{G}$ containing n trees, moreover as they are the only edges of weight $-\infty$ w.r.t. $\hat{w}'$ we must have $Z \subseteq E(M)$ and $|E(M') \setminus Z| = n - 1$. Let M be a MST of G w.r.t. w' and $e = (u, v)$ be an edge in $E(M') \setminus Z$, we now show that $\mu(e)$ must belong to M. Let C' be the cut-set containing the edges of $\hat{G}$ whose endpoints lie in two different components of $M - \{e\}$. Clearly e is the lightest among the edges in C', and since $\hat{w}'(e) > -\infty$ we have that $C' \cap Z = \emptyset$. Let $C = \{\mu(f) : f \in C'\}$ and notice that C is a cut-set for G. Since $\hat{w}'(f) = w'(\mu(f)) \forall f \in C$, it follows that $\mu(e)$ is also the lightest edge in C

and hence it belongs to M. The claim follows from the fact that, by construction, the function μ is injective. □

Once the maximum degree of the tree has been reduced to a constant, the query time of our oracle becomes $O(k^2 \log^2 n)$. Thus, we have finally proven the following result:

Theorem 1 *Given a MST T of a real-weighted n-vertex and m-edge graph G, there exists a sensitivity oracle for T of size $O(m \log^2 n)$. The oracle can be built in $O(m \log^3 n)$ time and, for any batch consisting of an arbitrary number k of modifications to the edges of G, it is able to report in $O(k^2 \log^2 n)$ time the (at most $2k$) edges either exiting from or entering into the MST.*

An immediate consequence of this result is that the additional query $q(u, v)$ can then be answered in $O(k^2 \log^2 n)$ time.

3 An f-EFT $(2|F| + 1)$-ASPT and a Corresponding Oracle

In this section we show how to compute an f-EFT $(2|F| + 1)$-ASPT H of G, i.e., a subgraph H of G which contains an SPT of G, and such that $d_{H-F}(s, v) \leq (2|F| + 1)d_{G-F}(s, v)$ for every $v \in V(G)$ and for every subset $F \subseteq E(G)$ of at most f failed edges.

Since up to f edges can fail, we can observe that whenever G is $(f + 1)$-edge-connected, H must contain $\Omega(fn)$ edges even if we are only interested in preserving the connectivity of G. Indeed, the degree of each vertex v in H must be at least $f + 1$ since otherwise v could become disconnected in $H - F$. Here we show that $|E(H)| = O(fn)$ edges also suffice if we aim to preserve distances that are at most $(2f + 1)$-stretched w.r.t. the surviving part of G.

Let $d_X(u, u')$ and $\pi_X(u, u')$ denote the distance and the shortest path between nodes u and u' in any subgraph X of G, respectively. When $u = s$, we will simply write $d_X(u')$ and $\pi_X(u')$. If π is a path, $\pi[u, u']$ will denote the subpath of π between $u, u' \in V(\pi)$.

For any given integer f, Algorithm 2 returns an f-EFT $(2|F| + 1)$-ASPT of G. First, it computes a SPT T of G that is used to assign a weight to the edges of an auxiliary graph $G' = (V(G), E(G), w')$. More precisely, the weight of an edge e of G' is 0 if e is also in T, otherwise it is equal to the sum of the corresponding edge weight in G and the distances in T between s and the endpoints of e. Then, $f + 1$ MSFs $M_0, \ldots, M_f$ of G' are iteratively computed: when we compute the i-th forest, we remove its edges M_i from G' before computing the $(i + 1)$-th forest, so that the sets M_i are pairwise disjoint. The sought subgraph H contains all the edges of the sets M_i, and has therefore size $O(fn)$. Notice that M_0 coincides with $E(T)$.

We now argue that H is indeed an f-EFT $(2|F|+1)$-ASPT of G. Our proof strategy is as follows: we first show that, when the weight function w' is considered, any MSF of $H - F$ is also an MFS of $G - F$ (Lemma 2). This allows us to upper bound the length of a shortest-path π' from s to t in $H - F$ as a function of $d_G(t)$ and the weights $w'(e')$ of the edges e' in π' (Lemma 3). Next, Lemma 4 bounds each of these

Algorithm 2: Algorithm for computing an f-EFT $(2|F|+1)$-ASPT of G.

```
1  $T \leftarrow$ compute a SPT of $G$
2  for $(u,v) \in E(G)$ do
3      if $(u,v) \in E(T)$ then $w'(u,v) \leftarrow 0$ else $w'(u,v) \leftarrow d_T(u) + w(u,v) + d_T(v)$
4  $G' \leftarrow (V(G), E(G), w')$
5  $G_0 \leftarrow G'$
6  for $i = 0, \ldots, f$ do
7      $M_i \leftarrow$ edges of a MSF of $G_i$ (w.r.t. $w'$)
8      $G_{i+1} \leftarrow G_i - M_i$
9  $H \leftarrow$ subgraph of $G$ containing the edges in $\bigcup_{i=0}^{f} M_i$
10 return $H$
```

weights $w'(e')$ by the largest edge weight $w'(e)$ of some edge e in $\pi_{G-F}(t)$. Finally, our definition of w' provides the desired bound on the length of π' (see Lemma 5).

Fix a vertex t and let $\pi = \pi_{G-F}(t)$ be the shortest path from s to t in the surviving graph $G - F$.[10] The path π contains (a subset of) the vertices from several trees in the forest $T - F$. We say that an edge is *new* if its endpoints belong to two different trees in $T - F$. Let N be the set of new edges in π.

Now consider a MSF M of the graph $H - F$ (w.r.t. w'). This is also a MSF of the graph $G' - F$ (w.r.t. w') as shown by the following lemma.

Lemma 2 *For every $F \subseteq E(G)$ with $|F| \leq f$, any MSF M of $H - F$ (w.r.t. w') is also a MSF of $G' - F$ (w.r.t. w').*

Proof In what follows, whenever ties arise we break them by prioritizing the edges in H. First we show that, given any cut-set[11] C of G', H contains the $\min\{|C|, f+1\}$ lightest edges of C. Indeed, for any set M_i, consider the set $C_i = C \setminus \cup_{j=0}^{i-1} M_j$. Either C_i is non empty, and therefore M_i contains the lightest edge in C_i, or $C_i = \emptyset$ which means that each edge in C belongs to some set M_j and hence to H.

Let M' be a MSF of $G' - F$. We prove the claim by showing that each edge $e \in E(M')$ must also belong to M. Let C' be the cut-set of G' that contains e and every edge $e' \in E(G')$ that forms a cycle with e in $M' \cup \{e'\}$. Since e is the lightest edge of $C' \setminus F$, it is within the $f+1$ lightest edges of C'. As a consequence $e \in E(H - F)$, and it also belongs to M as it is the lightest edge in $C' \cap E(H - F)$. □

Let $\pi' = \pi_M(s,t)$ and notice that π' traverses each tree of the forest $T - F$ at most once since the edges in $E(T)$ have weight 0 in H. Once again, let N' be the set of new edges of π'. By using the path π', we now provide an upper bound to the distance $d_{H-F}(t)$:

Lemma 3 $d_{H-F}(t) \leq w(\pi') \leq \sum_{e \in N'} w'(e) + d_G(t)$.

[10] We assume that such a path exists, as otherwise $d_{G-F}(t) = +\infty$ which implies $d_{H-F}(t) = +\infty$, and we are done.

[11] A cut-set of a graph X is a subset of $E(X)$ whose removal increases the number of connected components of X.

Proof Let M be a MSF of the graph $H - F$ (w.r.t. w'). The first inequality is trivial as $\pi' = \pi_M(s, t)$ is a path (not necessarily shortest) between s and t in (a subgraph of) $H - F$, hence we focus on proving the second inequality.

Let $T_0, \ldots, T_h$ be the trees of $T - F$ traversed by π', in order, and let $e'_i = (v_{i-1}, u_i)$ be the new edge in π' connecting a vertex v_{i-1} of T_{i-1} to a vertex u_i of T_i. Thus, we have $N' = \{e'_1, \ldots, e'_h\}$. We call r_i the vertex in $V(T_i) \cap V(\pi')$ that has the lowest depth in T_i.[12]

According to this definition, r_0 coincides with s, r_h is the LCA in T_h between u_h and t, and r_i is the LCA in T_i between u_i and v_i, for every $0 < i < h$.

We prove by induction on i that $w(\pi'[s, r_i]) \le \sum_{j=1}^{i} w'(e'_j)$. The base case $i = 0$ is trivially true. Now suppose that the inductive hypothesis holds for i, we prove it also for $i + 1$:

$$\begin{aligned} w(\pi'[s, r_{i+1}]) &= w(\pi'[s, r_i]) + d_{T_i}(r_i, v_i) + w(e'_{i+1}) + d_{T_{i+1}}(u_{i+1}, r_{i+1}) \\ &\le \sum_{j=1}^{i} w'(e'_j) + d_T(v_i) + w(e'_{i+1}) + d_T(u_{i+1}) \\ &\le \sum_{j=1}^{i} w'(e'_j) + w'(e'_{i+1}) = \sum_{j=1}^{i+1} w'(e'_j). \end{aligned}$$

We now use the fact that $d_{T_h}(r_h, t) = d_T(r_h, t) = d_G(r_h, t)$ to prove the claim:

$$w(\pi') = w(\pi'[s, r_h]) + w(\pi'[r_h, t]) \le \sum_{j=1}^{h} w'(e'_j) + d_{T_h}(r_h, t) \le \sum_{j=1}^{h} w'(e'_j) + d_G(t).$$

□

Next lemma shows that the weights of the new edges of π' are, in turn, upper bounded by the weight of some new edge of the path π.

Lemma 4 *For each $e' \in N'$, we have $w'(e') \le \max_{e \in N} w'(e)$.*

Proof Let $e' = (x, y)$ be an arbitrary edge in N'. W.l.o.g., we assume that the path π' traverses the vertices s, x, y, t in this order. We recall that the path π' traverses each tree in $T - F$ at most once, i.e., all the vertices of π' that belong to the same tree in $T - F$ must be contiguous in π'. Moreover, as e' is new, x and y belong to two different trees in $T - F$.

Let Z be the set of trees of the forest $T - F$ that are traversed by the path π. Let u' be the last vertex of $\pi'[s, x]$ that belongs to a tree, say T_u, in Z (see Fig. 4). Observe that u' is always defined since s belongs to some tree of Z. In a similar way, let v' be the first vertex of $\pi'[y, t]$ that belongs to a tree, say T_v, in Z. Again, observe that v' is always defined as t belongs to some tree of Z other than that containing s, hence $T_u \neq T_v$, and finally notice that $e' \in E(\pi'[u', v'])$. By our choice of T_u and T_v, we

[12] We think of T_i as rooted in the vertex of $V(T_i)$ which is closest to s in T.

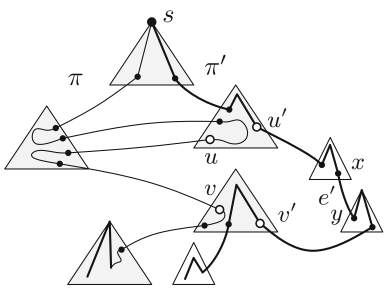


Fig. 4 The forest $T - F$ obtained by deleting the failed edges in F from T. The path π is the shortest path between s and t in $G - F$, while π' (in bold) is the unique path in M between the same vertices. Gray trees contain a vertex of π and are therefore in Z. Edges having endpoints in different trees are *new*

know that π encounters both a vertex of T_u and a vertex of T_v (in some order), so we let π^* be the minimal (w.r.t. inclusion) subpath of π with one endpoint, say u, in $V(T_u)$, and the other endpoint, say v, in $V(T_v)$.

Let $N^* = E(\pi^*) \cap N$ be the set of new edges in π^*. Notice that $N^* \neq \emptyset$ as $T_u \neq T_v$, and that adding the edges in N^* (weighted according to w') to M forms (at least) one cycle C containing both e' and an edge in N^*, say e^*. Since M is a MSF of $G' - F$, as shown by Lemma 2, we have that $w'(e') \le w'(e^*) \le \max_{e \in N} w'(e)$. □

Finally, next lemma relates the weights w' of the new edges of π to distances in the surviving graph $G - F$.

Lemma 5 *For $e \in N$, $w'(e) \le 2d_{G-F}(t)$.*

Proof Let $e = (u, v)$ with $d_{G-F}(u) \le d_{G-F}(v)$. Since e lies on the shortest path $\pi = \pi_{G-F}(s, t)$, we can write:

$$w'(e) = d_T(u) + w(e) + d_T(v) \le d_{G-F}(v) + d_T(v) \le 2d_{G-F}(v) \le 2d_{G-F}(t).$$

□

We are now ready to prove the main result of this section:

Theorem 2 *The graph H returned by* Algorithm 2 *is an f-EFT $(2|F| + 1)$-ASPT of G of size $O(fn)$. Moreover,* Algorithm 2 *requires $O(fm\,\alpha(m, n))$ time and $O(m)$ space.*

Proof First, observe that $\pi' = \pi_M(s, t)$ contains at most $|F|$ new edges. Indeed all the edges in $T - F$ have weight 0, while the remaining edges have a positive weight. This means that $E(T - F) \subseteq E(M)$. As $T - F$ has no more than $|F| + 1$ connected components, we have that at most $|F|$ other edges—in addition to the ones in $E(T - F)$—can belong to M.

By using the above fact in conjunction with Lemmas 3–5, we can write:

$$\begin{aligned} d_{H-F}(t) &\le w(\pi') \le \sum_{e \in N'} w'(e) + d_G(t) \le |F| \max_{e \in N'} w'(e) + d_G(t) \\ &\le |F| \max_{e \in N} w'(e) + d_G(t) \le 2|F| d_{G-F}(t) + d_{G-F}(t) \\ &= (2|F| + 1) d_{G-F}(t). \end{aligned} \tag{1}$$

We recall that this holds for every vertex $t \in V(G)$. Concerning the computational complexity of Algorithm 2, we make use of Chazelle's algorithm [17]—that computes a MSF in $O(m\,\alpha(m,n))$ time and linear space—to compute the $f+1$ MSFs $M_0, \dots, M_f$. □

3.1 A Corresponding Path/Distance Reporting Oracle

In this section we show how to build an oracle that, given a positively real-weighted graph G and a distinguished source vertex s, is able to answer queries of the form:

"Given a set F of at most f edge failures in G, and a destination node t in G, report a $(2|F|+1)$-approximate path/distance from s to t in $G - F$."

We first compute a SPT T of G and an f-EFT $(2|F|+1)$-ASPT H of G, as shown in the previous section. Then, the oracle is composed of three ingredients:

- The tree T and all the distances $d_T(v) = d_G(v)$ from s to any vertex $v \in V(G)$;
- A MSF sensitivity oracle Q for T w.r.t. H with weights w', built as shown in Sect. 2. According to Theorem 1, this oracle has size $O(fn\log^2 n)$ and can be built in $O(fn\log^3 n)$ time;
- An oracle to answer LCA queries between two vertices in T. Such an oracle can be built in linear time and has a constant query time, as shown in [32].

The resulting size is therefore $O(fn\log^2 n)$, and the time required to build our oracle is $O(fm\,\alpha(m,n) + fn\log^3 n)$. Interestingly, if we do not fix in advance the value of f, we can build in $O(m\log^3 n)$ time an oracle of size $O(m\log^2 n)$ that is able to report $(2|F|+1)$ approximate paths/distances, for *any* number $|F|$ of faults. Indeed, the number of edges in H is upper bounded by m, i.e., the number of edges of G.

In the following, we analyze the query time of the oracle, by first facing a path-reporting query, and then a simpler distance query.

Answering a Path-Reporting Query

To return a $(2|F|+1)$-approximate path between s and t, it suffices to report the path $\pi' = \pi_M(s,t)$, as shown by Eq. (1).

To this aim, we first query the MSF oracle Q through the additional query $q(s,t)$, which will return all the new edges on the unique path from s to t in the updated MSF. Let $\langle e'_1, \dots, e'_h \rangle$ be these new edges, in order, with $e'_i = (v_{i-1}, u_i)$. For $0 < i < h$, let r_i be the LCA in T_i between u_i and v_i, and let r_h be the LCA in T_h between u_h and t. We now have all the pieces needed to reconstruct and return the path π'. Indeed, if we let $\pi'_i = \pi_T(u_i, r_i) \circ \pi_T(r_i, v_i)$, the following holds:

$$\pi' = \pi_T(s, v_0) \circ e'_1 \circ \pi'_1 \circ e'_2 \circ \pi'_2 \circ \cdots \circ \pi'_{h-1} \circ e'_h \circ \pi_T(u_h, r_h) \circ \pi_T(r_h, t) \quad (2)$$

where each subpath is entirely in T and all the endpoints are known. The whole procedure requires $O(|F|^2 \log^2 n)$ time to perform the query $q(s,t)$ on Q (as observed at the end of Sect. 2), $O(|F|)$ time for the LCA queries, and $O(|\pi'|)$ time to reconstruct the path. The overall query time is therefore $O(|F|^2\log^2 n + |\pi'|)$.

Answering a Distance Query To report the length of a $(2|F|+1)$-approximate path from s to t, we can replace each subpath in Eq. (2) with the corresponding distance, in order to obtain:

$$w(\pi') = d_T(v_0) + \sum_{i=1}^{h-1} \left(w(e'_i) + d_T(u_i, r_i) + d_T(r_i, v_i) + w(e'_{i+1}) \right) \\ + d_T(u_h, r_h) + d_T(r_h, t).$$

The above quantity can be computed in $O(h) = O(|F|)$ time, once we know the edges $e_1, \ldots, e_h$ and we notice that $w(e'_i) = w'(e'_i) - d_T(v_{i-1}) - d_T(u_i)$, and moreover that if x is a descendant of r_i in T, then $d_T(r_i, x) = d_T(x) - d_T(r_i)$. The overall query time is thus $O(|F|^2 \log^2 n)$.

Summarizing, we can give the following:

Theorem 3 *Given a positively real-weighted n-vertex and m-edge graph G, and given a distinguished source vertex s of G, there exists an f-EFT SSDO of G that can be built in $O(fm\,\alpha(m,n) + fn\log^3 n)$ time. The oracle has size $O(fn\log^2 n)$ and, for any set F of failures, it is able to report $(2|F|-1)$-approximate distances (resp., approximate shortest paths) from s in $G - F$ in $O(|F|^2 \log^2 n)$ time (resp., $O(|F|^2 \log^2 n)$ time plus the size of the reported path).*

4 A Lower Bound to the Size of a (log n)-EFT σ-ASPT

In this section we show that, if the long-standing girth conjecture of Erdős [26] is true, then, for any integer $k \geq 1$, any f-EFT σ-ASPT with $f \geq \log n$ and $\sigma < \frac{3k+1}{k+1}$ requires $\Omega(n^{1+\frac{1}{k}})$ edges. In particular, this implies that if we want to build a structure which is resistant to at least $\log n$ edge failures and has stretch less than 2, then it must contain $\Theta(n^2)$ edges. Very recently, in [5], Baswana et al. showed that a similar construction implies a lower bound of $\Omega(2^f n)$ edges for any structure preserving connectivity on *directed* graphs when up to f edges can fail.

Theorem 4 *There are graphs G for which any f-EFT σ-ASPT with $f \geq \log n$ and $\sigma < \frac{3k+1}{k+1}$ contains $\Omega(n^{1+\frac{1}{k}})$ edges.*

Proof Let G be a graph on η vertices with girth $g = 2k+2$ and $\Omega(\eta^{1+\frac{1}{k}})$ edges (according to the girth conjecture, such a graph always exists). We construct a weighted graph G' in the following way (see Fig. 5): we add to G a binary tree T rooted in s with η leaves and height $h = \lceil \log \eta \rceil$, and we further add an edge from each leaf of T to a distinct vertex of $V(G)$, in an arbitrary way. The weights of $E(G)$ and $E(T)$ will be set to 1 and 0, respectively, while the remaining additional edges will have weight $x = \frac{g}{2} - 1$. Observe that the total number of vertices of G' is $n = 3\eta - 1$, hence $|E(G')| = \Omega(n^{1+\frac{1}{k}})$.

Let H be any f-EFT σ-ASPT of G' rooted in s, with $f \geq \log n$ and $\sigma < \frac{3k+1}{k+1}$. We will show that H must contain all the edges of $E(G)$. Indeed, suppose that an edge $e = (u, v) \in E(G)$ is missing from H, and let u' be the unique leaf of the T such that $(u, u') \in E(G')$. We let $\langle s = u_0, u_1, \ldots, u_k \rangle$ be the sequence of internal vertices

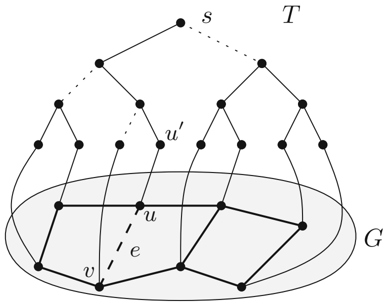


Fig. 5 Graph G' used in the lower bound construction. The dashed edge e does not belong to H while the dotted edges belong to F. Bold edges have weight 1, tree edges have weight 0, and the remaining edges—connecting the leaves of T to the vertices in G—have weight x

of T traversed by $\pi_T(s, u')$, and let e_i be the edge incident to u_i other than that in $E(\pi_T(s, u'))$. We choose $F = \{e_0, e_1, \dots, e_k\}$ as shown in Fig. 5. It is easy to see that $|F| \leq h = \lceil \log \eta \rceil \leq \log \eta + 1 \leq \log n$, and that each path from s to any vertex of $V(G)$ in $G' - F$ has the path $\pi = \pi_T(s, u') \circ (u', u)$ as a prefix, hence the same must hold in $H - F$. Therefore, we know that $\pi_{H-F}(s, v) = \pi \circ \pi_{H-F}(u, v)$. Observe that either $\pi_{H-F}(u, v)$ passes through a vertex in $V(T)$ or not. In the former case, it must contain at least an edge of weight 1 and two edges of weight x, hence $w(\pi_{H-F}(u, v)) \geq 2x + 1 = g - 1$. Otherwise, since the girth of G is g, $w(\pi_{H-F}(u, v)) \geq g - 1$. In both cases we have that $d_{H-F}(v) = w(\pi_{H-F}(s, v)) = w(\pi) + w(\pi_{H-F}(u, v)) \geq \frac{g}{2} - 1 + g - 1 = \frac{3}{2}g - 2$. At the same time, it holds $d_{G'-F}(v) = w(\pi \circ (u, v)) = \frac{g}{2}$. This implies that the stretch factor of H would be at least $3 - \frac{4}{g} = \frac{3k+1}{k+1}$, a contradiction. □

5 Conclusions and Open Problems

In this paper we have shown how to build an f-EFT $(2|F| + 1)$-ASPT of size $O(fn)$, thus providing a substantial improvement over the previously known structure of [42]. Such a structure has an (asymptotically) optimal size, as $\Omega(fn)$ edges are needed even to preserve connectivity between the source vertex and the other vertices of the graph when f edge failures can happen. It is not clear, however, whether the stretch can be improved without increasing the size of the structure. The lower bound of $\Omega(n^{1+\frac{1}{k}})$ edges to the size of any f-EFT ASPT shown in Sect. 4 only works for more than $\log n$ failures, and for a fixed size of $\Theta(fn)$ edges only implies a lower bound to the stretch of at most 3, hence we look at the problem of closing this gap as an interesting challenge. Nonetheless, we point out that this already shows that it is not possible to obtain a stretch arbitrarily close to 1 with a quasi-linear number edges, as it happens for the single-failure case. To this respect, it would be worthy to understand if the lower bound can be extended to work—even if in some weaker form—for a constant number of edge failures.

Furthermore, we have also shown how to convert our f-EFT $(2|F| + 1)$-ASPT into an f-EFT $(2|F| + 1)$-SSDO of size $O(fn \log^2 n)$ than can answer distance queries in time $O(|F|^2 \log^2 n)$. This oracle is also able to report the path corresponding to a query by using an additional time proportional to the number of edges of the path itself (i.e., constant time per edge). To obtain this latter result we developed a sensitivity oracle of size $O(m \log^2 n)$ that is able to report in $O(k^2 \log^2 n)$ time the changing edges of a MSF of a graph following the insertion/deletion/weight-update of k edges of the graph. This provides the current best worst-case solution to the fully-dynamic MSF problem for relatively short sequences of updates when the starting graph is arbitrary. Notice that a lower-bound of $\Omega(m)$ edges to the size of any MSF sensitivity oracle trivially holds when $k = \Theta(m)$, since storing the whole graph G is required. It would therefore be interesting to understand whether it is possible to shave off the polylogarithmic factors from the size of our constructions and from the query time. The latter improvement would also result in a query time that is a function of only the number of changing edges, i.e., it is constant as soon as $k = O(1)$.

Fudning The fund was grant by Ministero dell'Istruzione, dell'Università e della Ricerca, Algorithms for (fault-tolerant) pairwise spanners and distance oracles.

References

1. Ahmed, A.R., Bodwin, G., Sahneh, F.D., Hamm, K., Jebelli, M.J.L., Kobourov, S.G., Spence, R.: Graph spanners: a tutorial review. arXiv:1909.03152 (2019)
2. Alstrup, S., Holm, J., de Lichtenberg, K., Thorup, M.: Maintaining information in fully dynamic trees with top trees. ACM Trans. Algorithms **1**(2), 243–264 (2005). https://doi.org/10.1145/1103963.1103966
3. Althöfer, I., Das, G., Dobkin, D.P., Joseph, D., Soares, J.: On sparse spanners of weighted graphs. Discrete Comput. Geom. **9**, 81–100 (1993). https://doi.org/10.1007/BF02189308
4. Ausiello, G., Franciosa, P.G., Italiano, G.F., Ribichini, A.: On resilient graph spanners. Algorithmica **74**(4), 1363–1385 (2016). https://doi.org/10.1007/s00453-015-0006-x
5. Baswana, S., Choudhary, K., Roditty, L.: Fault-tolerant subgraph for single-source reachability: general and optimal. SIAM J. Comput. **47**(1), 80–95 (2018). https://doi.org/10.1137/16M1087643
6. Baswana, S., Khanna, N.: Approximate shortest paths avoiding a failed vertex: near optimal data structures for undirected unweighted graphs. Algorithmica **66**(1), 18–50 (2013). https://doi.org/10.1007/s00453-012-9621-y
7. Baswana, S., Telikepalli, K., Mehlhorn, K., Pettie, S.: Additive spanners and (alpha, beta)-spanners. ACM Trans. Algorithms **7**(1), 5 (2010). https://doi.org/10.1145/1868237.1868242
8. Bernstein, A., Karger, D.R.: A nearly optimal oracle for avoiding failed vertices and edges. In: STOC, pp. 101–110 (2009)
9. Bilò, D., Choudhary, K., Gualà, L., Leucci, S., Parter, M., Proietti, G.: Efficient oracles and routing schemes for replacement paths. In: 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, pp. 13:1–13:15 (2018). https://doi.org/10.4230/LIPIcs.STACS.2018.13
10. Bilò, D., Grandoni, F., Gualà, L., Leucci, S., Proietti, G.: Improved purely additive fault-tolerant spanners. In: ESA, pp. 167–178 (2015)
11. Bilò, D., Gualà, L., Leucci, S., Proietti, G.: Compact and Fast Sensitivity Oracles for Single-Source Distances. In: 24th Annual European Symposium on Algorithms (ESA 2016), *Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 57, pp. 13:1–13:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2016). https://doi.org/10.4230/LIPIcs.ESA.2016.13. http://drops.dagstuhl.de/opus/volltexte/2016/6364
12. Bilò, D., Gualà, L., Leucci, S., Proietti, G.: Fault-tolerant approximate shortest-path trees. Algorithmica **80**(12), 3437–3460 (2018). https://doi.org/10.1007/s00453-017-0396-z
13. Bodwin, G., Grandoni, F., Parter, M., Williams, V.V.: Preserving distances in very faulty graphs. In: 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, pp. 73:1–73:14 (2017). 10.4230/LIPIcs.ICALP.2017.73
14. Bodwin, G., Patel, S.: A trivial yet optimal solution to vertex fault tolerant spanners. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019., pp. 541–543 (2019). 10.1145/3293611.3331588

15. Braunschvig, G., Chechik, S., Peleg, D., Sealfon, A.: Fault tolerant additive and (μ, α)-spanners. Theor. Comput. Sci. **580**, 94–100 (2015). https://doi.org/10.1016/j.tcs.2015.02.036
16. Charalampopoulos, P., Mozes, S., Tebeka, B.: Exact distance oracles for planar graphs with failing vertices. In: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019, pp. 2110–2123 (2019). https://doi.org/10.1137/1.9781611975482.127
17. Chazelle, B.: A minimum spanning tree algorithm with inverse-ackermann type complexity. J. ACM **47**(6), 1028–1047 (2000). https://doi.org/10.1145/355541.355562
18. Chechik, S.: New additive spanners. In: SODA, pp. 498–512 (2013). https://doi.org/10.1137/1.9781611973105.36
19. Chechik, S.: Approximate distance oracles with constant query time. In: STOC, pp. 654–663 (2014). https://doi.org/10.1145/2591796.2591801
20. Chechik, S., Langberg, M., Peleg, D., Roditty, L.: f-sensitivity distance oracles and routing schemes. Algorithmica **63**(4), 861–882 (2012). https://doi.org/10.1007/s00453-011-9543-0
21. D'Andrea, A., D'Emidio, M., Frigioni, D., Leucci, S., Proietti, G.: Path-fault-tolerant approximate shortest-path trees. In: SIROCCO, pp. 224–238 (2015). https://doi.org/10.1007/978-3-319-25258-2_16
22. Duan, R., Pettie, S.: Dual-failure distance and connectivity oracles. In: SODA, pp. 506–515 (2009). http://dl.acm.org/citation.cfm?id=1496770.1496826
23. Elkin, M., Pettie, S.: A linear-size logarithmic stretch path-reporting distance oracle for general graphs. ACM Trans. Algorithms **12**(4), 50:1-50:31 (2016). https://doi.org/10.1145/2888397
24. Eppstein, D.: Offline algorithms for dynamic minimum spanning tree problems. J. Algorithms **17**(2), 237–250 (1994). https://doi.org/10.1006/jagm.1994.1033
25. Eppstein, D., Galil, Z., Italiano, G.F., Nissenzweig, A.: Sparsification: a technique for speeding up dynamic graph algorithms. J. ACM **44**(5), 669–696 (1997). https://doi.org/10.1145/265910.265914
26. Erdős, P.: Extremal problems in graph theory. In: Theory of Graphs and its Applications, pp. 29–36 (1964)
27. Frederickson, G.N.: Data structures for on-line updating of minimum spanning trees, with applications. SIAM J. Comput. **14**(4), 781–798 (1985). https://doi.org/10.1137/0214055
28. Grandoni, F., Williams, V.V.: Improved distance sensitivity oracles via fast single-source replacement paths. In: FOCS, pp. 748–757 (2012)
29. Gupta, M., Khan, S.: Multiple source dual fault tolerant BFS trees. In: 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland, pp. 127:1–127:15 (2017). https://doi.org/10.4230/LIPIcs.ICALP.2017.127
30. Gupta, M., Singh, A.: Generic single edge fault tolerant exact distance oracle. In: 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic, pp. 72:1–72:15 (2018). https://doi.org/10.4230/LIPIcs.ICALP.2018.72
31. Hagerup, T., Miltersen, P.B., Pagh, R.: Deterministic dictionaries. J. Algorithms **41**(1), 69–85 (2001). https://doi.org/10.1006/jagm.2001.1171
32. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**(2), 338–355 (1984). https://doi.org/10.1137/0213024
33. Henzinger, M.R., King, V.: Maintaining minimum spanning forests in dynamic graphs. SIAM J. Comput. **31**(2), 364–374 (2001). https://doi.org/10.1137/S0097539797327209
34. Holm, J., de Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM **48**(4), 723–760 (2001). https://doi.org/10.1145/502090.502095
35. Jordan, C.: Sur les assemblages de lignes. (1869)
36. Knuth, D.E.: The art of computer programming, Volume I: Fundamental Algorithms, 3rd Edition. Addison-Wesley (1997). https://www.worldcat.org/oclc/312910844
37. Nardelli, E., Proietti, G., Widmayer, P.: Swapping a failing edge of a single source shortest paths tree is good and fast. Algorithmica **35**(1), 56–74 (2003)
38. Parter, M.: Dual failure resilient BFS structure. In: PODC, pp. 481–490 (2015)
39. Parter, M.: Fault-tolerant logical network structures. Bulletin of the EATCS **118** (2016). http://eatcs.org/beatcs/index.php/beatcs/article/view/403
40. Parter, M.: Vertex fault tolerant additive spanners. Distrib. Comput. **30**(5), 357–372 (2017). https://doi.org/10.1007/s00446-015-0252-9

41. Parter, M., Peleg, D.: Sparse fault-tolerant BFS structures. ACM Trans. Algorithms **13**(1), 11:1-11:24 (2016). https://doi.org/10.1145/2976741
42. Parter, M., Peleg, D.: Fault-tolerant approximate BFS structures. ACM Trans. Algorithms **14**(1), 10:1-10:15 (2018). https://doi.org/10.1145/3022730
43. Sanders, P., Mehlhorn, K., Dietzfelbinger, M., Dementiev, R.: Sequential and parallel algorithms and data structures: the basic toolbox. Springer (2019). https://doi.org/10.1007/978-3-030-25209-0
44. Thorup, M., Zwick, U.: Approximate distance oracles. J. ACM **52**(1), 1–24 (2005). https://doi.org/10.1145/1044731.1044732

Authors and Affiliations

Davide Bilò[1] · Luciano Gualà[2] · Stefano Leucci[3] · Guido Proietti[3,4]

✉ Stefano Leucci
stefano.leucci@univaq.it

Davide Bilò
davide.bilo@uniss.it

Luciano Gualà
guala@mat.uniroma2.it

Guido Proietti
guido.proietti@univaq.it

1 Dipartimento di Scienze Umanistiche e Sociali, Università di Sassari, Sassari, Italy

2 Dipartimento di Ingegneria dell'Impresa, Università di Roma "Tor Vergata", Rome, Italy

3 Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università degli Studi dell'Aquila, L'Aquila, Italy

4 Istituto di Analisi dei Sistemi ed Informatica "A. Ruberti", CNR, Rome, Italy