Compact and Fast Sensitivity Oracles for Single-Source Distances*

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

1 **DUMAS, Università di Sassari, Sassari, Italy**
davide.bilo@uniss.it
2 **DII, Università di Roma "Tor Vergata", Rome, Italy**
guala@mat.uniroma2.it
3 **DI, "Sapienza" Università di Roma, Rome, Italy**
leucci@di.uniroma1.it
4 **DISIM, Università degli Studi dell'Aquila, L'Aquila, Italy**
IASI, CNR, Roma, Italy
guido.proietti@univaq.it

Abstract

Let s denote a distinguished source vertex of a non-negatively real weighted and undirected graph G with n vertices and m edges. In this paper we present two efficient *single-source approximate-distance sensitivity oracles*, namely *compact* data structures which are able to *quickly* report an approximate (by a multiplicative stretch factor) distance from s to any node of G following the failure of any edge in G. More precisely, we first present a sensitivity oracle of size $O(n)$ which is able to report 2-approximate distances from the source in $O(1)$ time. Then, we further develop our construction by building, for any $0 < \varepsilon < 1$, another sensitivity oracle having size $O\left(n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$, and is able to report a $(1+\varepsilon)$-approximate distance from s to any vertex of G in $O\left(\log n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ time. Thus, this latter oracle is essentially optimal as far as size and stretch are concerned, and it only asks for a logarithmic query time. Finally, our results are complemented with a space lower bound for the related class of single-source *additively-stretched* sensitivity oracles, which is helpful to realize the hardness of designing compact oracles of this type.

1998 ACM Subject Classification G.2.2 [Graph Theory] Graph algorithms, Trees

Keywords and phrases fault-tolerant shortest-path tree, approximate distance, distance sensitivity oracle

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.13

1 Introduction

The term *distance oracle* was coined by Thorup and Zwick [19], to emphasize the quality of a data structure that, despite its sparseness, is able to report very quickly provably good approximate distances between any pair of nodes in a graph. Indeed, it is well-known that in huge graphs the trade-off between time and space for *exact* distance queries is a very critical issue: at its extremes, either we use a quadratic (unfeasible) space to reply in constant time, or we use a linear space to reply at an unsustainable large time. Thus, a wide body of literature focused on the problem of developing intermediate solutions in between these two opposite approaches, with the goal of designing more and more compact and fast oracles. This already complex task is further complicated as soon as edge or vertex failures enter

* A full version of the paper is available at http://arxiv.org/abs/1608.04769.

into play: here, the oracle should be able to return (approximate) distances following the failure of some component(s) in the underlying graph, or in other words to be *fault-tolerant*, thus introducing an additional overload to the problem complexity. This kind of oracle is also known as *distance sensitivity oracle*. In this paper we focus our attention on a such challenging scenario, but we restrict our attention to the prominent case in which concerned distances are from a fixed source only, which is of special interest in several network-based applications.

1.1 Related work

Let s denote a distinguished source vertex of a non-negatively real weighted and undirected n-vertex and m-edge graph $G = (V(G), E(G), w)$. For the sake of avoiding technicalities, we assume that G is 2-edge-connected, although this assumption can be easily relaxed without affecting our results. A *single-edge-fault-tolerant α-single-source distance oracle* (EFT α-SSDO in the following), with $\alpha \geq 1$, is a data structure that for any $v \in V(G)$ and any $e \in E(G)$ is able to return an estimate of the distance in $G - e$ (i.e., the graph G deprived by e) between s and v, say $d_{G-e}(s, v)$, within the range $[d_{G-e}(s, v), \alpha \cdot d_{G-e}(s, v)]$. The term α is a.k.a. the *stretch factor* of the oracle.

A natural counterpart of such an oracle is an EFT *α-approximate shortest-path tree* (α-ASPT), i.e., a subgraph of G which, besides a SPT of G rooted at s, contains α-stretched shortest paths from s after the failure of any edge e in G. Such a structure is also known as a *single-source EFT α-spanner*. In some sense, a SSDO aims to convert in an explicit form the distance information that a corresponding ASPT may retain just in an implicit form, similarly to the process of maintaining in an n-size array all the distances from the source induced by the paths of a corresponding SPT. However, such a conversion process is far to be trivial in general and should be accomplished carefully, since the exploitation of the implicit information may introduce a dilatation in the final size of the oracle.

While the study of sensitivity oracles for all-pairs distances started right after the first appearance of [19], the single-source case was faced only later. More precisely, in [10] it was first proven that if we aim at *exact* distances, then $\Theta(n^2)$ space may be needed, already for undirected graphs and single edge failures, and independently of the query time. Then, in [1] the authors build in $O(m \log n + n \log^2 n)$ time a *single-vertex-fault-tolerant* (VFT) 3-SSDO of size $O(n \log n)$ and with constant query time. In the same paper, for *unweighted* graphs and for any $\varepsilon > 0$, the authors build in $O(m\sqrt{n/\varepsilon})$ time a VFT $(1 + \varepsilon)$-SSDO of size $O(\frac{n}{\varepsilon^3} + n \log n)$ and with constant query time. Both oracles are *path reporting*, i.e., they are able to report the corresponding approximate shortest path from the source in time proportional to the path size. Moreover, as discussed in [5], in both oracles/spanners the log-term in the size can be removed if *edge* failures are considered, instead of vertex failures. Finally, they can easily be transformed into corresponding E/VFT ASPTs having a same size and stretch. As far as this latter result is concerned, this was improved in [5], where it was given, for any (even non-constant) $\varepsilon > 0$, an E/VFT $(1 + \varepsilon)$-ASPT of size $O(\frac{n \log n}{\varepsilon^2})$, without providing a corresponding oracle, though.

Summarizing, we therefore have the following state-of-the-art for EFT SSDOs: if we insist on having linear-size and constant query time, then a $(1 + \varepsilon)$-stretch can be obtained only for unweighted graphs, while for weighted graphs the best current stretch is 3. Actually, this latter value can be reduced only by either paying a quadratic size (by storing for every $e \in E(G)$, the explicit distances from s in $G - e$), or an almost linear size but a super-linear query time (by storing and then inspecting the structure provided in [5]). So, the main open question is the following: can we develop a good space-time trade-off (ideally, linear space and constant query time) by guaranteeing a stretch less than 3 (ideally, arbitrarily close to 1)? In this paper, we make significant progresses in this direction.

1.2 Our results

Our main result is, for any arbitrary small $\varepsilon > 0$, the construction in $O(mn + n^2 \log n)$ time and $O\left(m + n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ space of an EFT $(1+\varepsilon)$-SSDO having size $O\left(n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ and query time $O\left(\log n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$. Thus, when ε is constant w.r.t. n, we get close to the ideal situation we were depicting above: our oracle has linear space, stretch arbitrarily close to 1, and a logarithmic query time. Moreover, it is interesting to notice that size and query time have an almost linear dependency on $1/\varepsilon$.

To the best of our knowledge, this is the first EFT SSDO guaranteeing a $(1+\varepsilon)$-stretch factor on weighted graphs. Interestingly, our construction is not obtained by the EFT $(1+\varepsilon)$-ASPT of size $O(\frac{n \log n}{\varepsilon^2})$ given in [5], whose conversion to a same size-stretch trade-off oracle sounds very hard, and is instead based on a quite different approach. More precisely, to get our size and query time bounds, we select a subset of *landmark* nodes of G, and for each one of them we store $O\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ *exact* post-failure (for an appropriate set of failing edges) distances from s. Then, when an edge e fails and we want to retrieve an approximate distance from s towards a fixed destination node t, we efficiently select with the promised query time a pivotal landmark node that actually sits on a path in $G - e$ from s to t whose length is within the bound. Notice that such a path is not *explicitly* stored in our oracle, so unfortunately we cannot return it in a time proportional to its size (besides the query time). In other words, our oracle is not inherently path-reporting, an we leave this point as a challenging open problem.

To get the reader acquainted with our technique, we first develop in $O(mn + n^2 \log n)$ time and $O(m)$ space an EFT 2-SSDO of size $O(n)$ and constant query time. This result is of independent interest, since it is the first EFT SSDO with both optimal size and query time having a stretch better than the long-standing barrier of 3. In this other oracle, once again we select a subset of landmark nodes of G, but in this case, to get the promised stretch, we do not need to maintain explicitly any exact distances towards them. Rather, for the failure of an edge e of G and for a fixed destination node t, a structural property of 2-stretched post-failure paths will allow us to return the 2-approximate distance from s by simply understanding whether there exists a pivotal landmark node associated with t. Actually, we show that such an association can be established by formulating a corresponding *bottleneck vertex query* problem on a rooted tree, that can be answered in $O(1)$ time by using a linear-size efficient data structure developed in [9].

Finally, in order to better appreciate the quality of our former oracle, we provide a lower bound on the bit size of any EFT β-*additive* SSDO, i.e., an oracle which is able to report a distance from s following an edge failure which is exact unless an additive term β. Notice that for weighted graphs, as in our setting, it only makes sense that such a β is depending on the actual queried distance d. Notice also that our linear-size EFT $(1+\varepsilon)$-SSDO can be revised as an EFT $(\varepsilon \cdot d)$-additive SSDO. So, a naturally arising question is: for a given $0 < \delta \leq 1$, can we devise a compact EFT $(\varepsilon \cdot d^{1-\delta})$-additive SSDO? We provide an answer in the negative, by showing a class of graphs for which a corresponding set of oracles of this sort would contain at least an element of $\Omega(n^2)$ bit size, regardless of its query time. Due to space limitations, the proof of this latter result will be given in the full version of the paper.

1.3 Other related results

Besides the aforementioned related work on single-source distance sensitivity oracles, we mention some further papers on the topic. For directed graphs with integer positive edge weights bounded by M, in [12] the authors show how to build efficiently in $\widetilde{O}(Mn^{\omega})$ time a *randomized* EFT 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. As far as *multiple* edge failures are concerned, in [6], for the failure of any set $F \subseteq E(G)$ of at most f edges of G, the authors build in $O(fm\,\alpha(m,n) + fn\log^3 n)$ time an f-EFT $(2|F|+1)$-SSDO of size $O(\min\{m, fn\}\log^2 n)$, with a query time of $O(|F|^2\log^2 n)$, and that is also able to report the corresponding path in the same time plus the path size. Notice that this oracle is obtained by converting a corresponding single-source f-EFT spanner having size $O(fn)$ and a same stretch. Notice also that if one is willing to use $O(m\log^2 n)$ space, such oracle will be able to handle any number of edge failures (i.e., up to m). Recently in [8], 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. For this problem, they build in $O(n(m+f^2))$ time, a $(2k-1)(2|F|+1)$-SSDO of size $O(kn\,f^{1+1/k})$ and constant query time, where $|F|$ denotes the size of the actual failing path, and $k \geq 1$ is a parameter of choice. Moreover, for the special case of $f = 2$, they give an ad-hoc solution, i.e., a 3-SSDO that can be built in $O(nm + n^2\log n)$ time, has size $O(n\log n)$ and constant query time.

In the past, several other research efforts have been devoted to *all-pairs distance oracles* (APDO) tolerating single/multiple edge/vertex failures. Quite interestingly, here $\widetilde{O}(n^2)$-size exact-distance sensitivity oracles are instead known, as opposed to the $\Omega(n^2)$ lower bound for the single-source case. More precisely, in [4] the authors built (on directed graphs) in $\widetilde{O}(mn)$ time a 1-E/VFT 1-APDO of size $\widetilde{O}(n^2)$ and with query time $O(1)$. For two failures, in [11] the authors built, still 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 [7] the authors built, for any integer $k \geq 1$, an f-EFT $(8k-2)(f+1)$-APDO of size $O(fk\,n^{1+1/k}\log(nW))$, where W is the ratio of the maximum to the minimum edge weight in G, and with a query time of $\widetilde{O}(|F|\,\log\log d)$, where F is the actual set of failing edges, and d is the distance between the queried pair of nodes in $G - F$.

As we said before, the natural counterpart of distance sensitivity oracles are the fault-tolerant spanners. Due to space limitations, for this related topic we refer the reader to the discussion and the references provided in [6]. However, it is worth mentioning that there is a line of papers on EFT ASPTs [14, 15, 16, 17], that as we said are very close in spirit to EFT SSDOs.

Finally, we mention that there is a large body of literature concerned with the design of ordinary (i.e., fault-free) distance oracles, and an extensive recent survey on the topic is given in [18].

1.4 Notation

For two given vertices x and y of an edge weighted graph H, we denote by $\pi_H(x,y)$ a shortest path between x and y in H and we denote by $d_H(x,y)$ the total length of $\pi_H(x,y)$. For two given paths P and P' such that P is a path between x and y and P' is a path between y and z, we denote by $P \circ P'$ the path from x to z obtained by concatening P and P'.

Let T be an SPT of G rooted at s, and let $e = (u,v)$ be an edge of T. In the rest of the paper, we always assume that u is closer to s than v w.r.t. the number of hops in T. Furthermore, we denote by T_v the subtree of T rooted at v. Finally, for a vertex $t \in T_v$, we denote by $A(t,e) = V(\pi_T(v,t))$ the set of *living ancestors* of T, t included, contained in T_v.

2 The EFT 2-SSDO

In this section we describe our EFT 2-SSDO with linear size and constant query time. Some of the ideas we develop here will be used in the next section, where we provide our main result.

For the rest of the paper, let T be a fixed SPT of G rooted at s that is stored in our distance oracle. First of all, observe that if there is no edge failure or the edge that has failed is not contained in T, then, for any vertex t, our distance oracle can return the (exact) distance value $d_T(s,t)$ in constant time. This is the case also when the edge $e=(u,v)$ that has failed is contained in T, but t is not a vertex of T_v. Therefore, in the rest of this section, we describe only how our distance oracle computes an approximate distance from s to t in $G-e$ when the edge $e=(u,v)$ that has failed is contained in T and the vertex t is contained in the subtree T_v.

The following lemma describes a simple but still interesting property that we exploit as key ingredient in our oracle. Let $e=(u,v)$ be a failing edge, we define a special replacement path from s to t as follows: $P_e(t)=\pi_{G-e}(s,v)\circ\pi_G(v,t)$.

▶ **Lemma 1.** *Let $e=(u,v)$ be a failing edge and $t\in V(T_v)$. At least one of the following conditions holds: (i) $d_{G-e}(s,t)\le w(P_e(t))\le 2d_{G-e}(s,t)$, (ii) $d_{G-e}(s,t)<2d_G(s,t)$.*

Proof. We assume that (ii) is false (i.e., $d_{G-e}(s,t)\ge 2d_G(s,t)$) and we prove that (i) must hold. Indeed:

$$\begin{aligned} d_{G-e}(s,t)\le w(P_e(t)) &= d_{G-e}(s,v)+d_G(v,t)\le d_{G-e}(s,t)+d_{G-e}(v,t)+d_G(v,t)\\ &= d_{G-e}(s,t)+2d_G(v,t)\le d_{G-e}(s,t)+2d_G(s,t)\le 2d_{G-e}(s,t). \end{aligned}$$ ◀

Notice that the length of $P_e(t)$ is available in constant time once we store $O(n)$ distance values, namely $d_{G-e}(s,v)$ for each $e=(u,v)\in E(T)$. Hence, the challenge here is to understand when $w(P_e(t))$ provides a 2-approximation of the distance $d_{G-e}(s,t)$ and when we can instead return the value $2d_G(s,t)\le 2d_{G-e}(s,t)$ (observe that $2d_G(s,t)$ could be in general smaller than $d_{G-e}(s,t)$). The idea of our oracle is that of selecting a subset of *marked* vertices for which this information can be stored and retrieved efficiently and from which we can derive the same information for the other nodes.

To this aim, we now describe an algorithm that preprocesses the graph and collects compact information that we will use later to efficiently answer distance queries. Consider the edges of T as traversed by a preorder visit from s. We define a total order relation $\prec$ on $E(T)$ as follows: we say that $e'\prec e''$ iff e' is traversed before e''. We also use $e'\preceq e''$ to denote that either $e'\prec e''$ or $e'=e''$.

Algorithm 1 considers the failing edges $e\in E(T)$ in preorder and computes a label $\ell(v)$ for each vertex $v\in V(G)$. This value will be either ∞ or a suitable edge $e\in E(T)$. Here we treat ∞ as a special label that satisfies $e'\prec\infty$ for every edge $e'\in E(T)$. We say that v is *marked* if $\ell(v)\ne\infty$, and we say that v is *marked at time* e if $\ell(v)\preceq e$. Intuitively, $\ell(v)$ is the time at which v first becomes marked.

More precisely, for each failing edge e, Algorithm 1, marks a vertex $t\in V(T_v)$ (at time e) iff vertex t fails two tests: the *distance test* and the *ancestor test*. In the distance test we check whether the path $P_e(t)$ suffices to provide a 2-stretched distance to t, while in the ancestor test we check whether a living ancestor of t has already been marked. Notice that the ancestor test guarantees that each vertex t is marked at most once during the whole execution of the algorithm (since $t\in A(t,e)$ by definition).

As a simple consequence of the above algorithm, we have:

Algorithm 1: Mark-up algorithm

```
1  for $v \in V$ do
2      $\ell(v) \leftarrow \infty$
3  for $e = (u,v) \in E(T)$ in preorder w.r.t. $T$ do
4      for $t \in V(T_v)$ in preorder w.r.t. $T$ do
5          if $w(P_e(t)) \le 2d_{G-e}(s,t)$ then                  // Distance test
6              do nothing
7          else if $\exists z \in A(t,e) : \ell(z) \neq \infty$ then    // Ancestor test
8              do nothing
9          else                                                  // Both tests failed
10             $\ell(t) \leftarrow e$                                   // Mark $t$ at time $e$
```

▶ **Lemma 2.** *Let $e \in E(T)$ be a failing edge and let t be a vertex such that $\ell(t) = e$, we have $d_{G-e}(s,t) < 2d_G(s,t)$.*

Proof. Since t is first marked at time e, it must have failed the distance test, i.e., $w(P_e(t)) > 2d_{G-e}(s,t)$. This means that condition (i) of Lemma 1 is false and hence condition (ii) must hold. ◀

Another useful property of the marked vertices is the following:

▶ **Lemma 3.** *Let $e \in E(T)$ be a failing edge and let t be a vertex such that $\ell(t) = e$, then $\pi_{G-e}(s,t)$ and $\pi_T(v,t)$ are edge disjoint.*

Proof. Let $e = (u,v)$ and assume by contradiction that $\pi_{G-e}(s,t)$ and $\pi_T(v,t)$ are not edge disjoint. Let (z,z') be an edge belonging to both paths, with z closer to v than z'. Notice that both z and z' are living ancestors of t, and that $z \neq t$.

Since t is first marked at time e, it must have failed the ancestor test. This implies that no other living ancestor of t is marked at time e. Moreover, as z is visited by the algorithm before t, it must have failed the ancestor test as well. Since z it is not marked at time e, it follows that it must have passed the distance test, i.e., $w(P_e(z)) \le 2d_{G-e}(s,z)$. We have $P_e(t) = P_e(z) \circ \pi_G(z,t)$ and hence:

$$\begin{aligned} w(P_e(t)) = w(P_e(z)) + d_G(z,t) &\le 2d_{G-e}(s,z) + d_G(z,t) \\ &\le 2d_{G-e}(s,z) + 2d_{G-e}(z,t) = 2d_{G-e}(s,t) \end{aligned}$$

which implies that t has passed the distance test and contradicts the hypothesis $\ell(t) = e$. ◀

The next lemma is the last ingredient of our oracle, and allows to distinguish the two cases of Lemma 1.

▶ **Lemma 4.** *Let $e = (u,v) \in E(T)$ be a failing edge and let $t \in V(T_v)$. If there exists $z \in A(t,e)$ such that $\ell(z) \preceq e$, then $d_{G-e}(s,t) \le 2d_G(s,t)$. If no such vertex z exists, then $d_{G-e}(s,t) \le w(P_e(t)) \le 2d_{G-e}(s,t)$.*

Proof. Let z be any vertex in $A(t,e)$ such that $\ell(z) \preceq e$, and let $e' = \ell(z)$. By the definition of living ancestor and by Lemma 3 we have that $\pi_{G-e'}(s,z)$ does not use the edge e (see

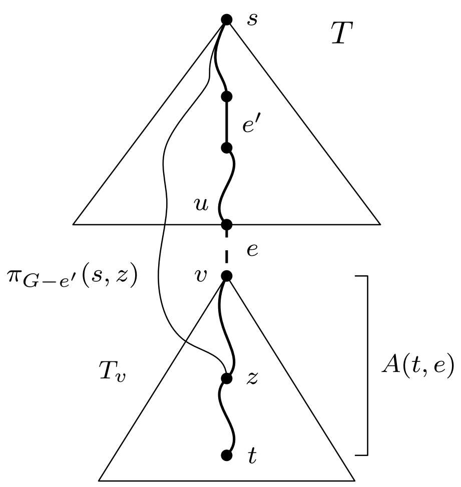

Figure 1 Representation of the proof of Lemma 4. The shortest path between s and t in T is shown in bold while the failing edge e is dashed. Notice that the path $\pi_{G-e'}(s,z)$ is edge disjoint from the path $\pi_T(v,z)$.

Figure 1). Since z is marked at time e' we have $d_{G-e'}(s,z) < 2d_G(s,z)$ (see Lemma 2). Thus, we have that $d_{G-e}(s,z) \leq w(\pi_{G-e'}(s,z)) = d_{G-e'}(s,z) < 2d_G(s,z)$. Therefore:

$$d_{G-e}(s,t) \leq d_{G-e}(s,z)+d_G(z,t) \leq 2d_G(s,z)+d_G(z,t) \leq 2d_G(s,z)+2d_G(z,t) = 2d_G(s,t).$$

If no such vertex z exists, then when Algorithm 1 considered edge e, the vertex z failed the ancestor test. Since t is not marked at time e (as otherwise we could choose $z = t$) it must have passed the distance test, i.e., $w(P_e(t)) \leq 2d_{G-e}(s,t)$. ◀

This latter lemma is exactly what we need in order to implement the query operation of our oracle. When edge $e = (u,v)$ is failing and we are queried for the distance of a vertex t, we first test whether $e \in E(T)$ and $t \in V(T_v)$: if the test fails we return the original distance $d_G(s,t)$.[1] If the test succeeds, we look for a vertex $z \in A(t,e)$ such that $\ell(z) \preceq e$. If such a vertex exists we return $2d_G(s,t)$, otherwise we return $w(P_e(t))$. Observe that in both cases we return a feasible 2-approximation of the distance $d_{G-e}(s,t)$.

In the following we will show how it is possible to determine in constant time whether such a vertex z exists. More precisely we only need to look for a vertex $x \in A(t,e)$ minimizing $\ell(x)$. If such a vertex satisfies $\ell(x) \preceq e$ then $z = x$ and we are done. On the converse, if $e \prec \ell(x)$, then we know that no vertex $z \in A(t,e)$ with $\ell(z) \preceq e$ can exist.

To this aim, we use a data structure for the *bottleneck vertex query* problem on trees (`BVQ` for short). In the `BVQ` problem we want to preprocess a vertex-weighted tree $\mathcal{T}$ in order to answer queries of this form: given two vertices $x, y \in V(\mathcal{T})$ report the lightest vertex on the (unique) path between x and y in $\mathcal{T}$. In [9], the authors show how to build, in $O(|V(\mathcal{T})| \log |V(\mathcal{T})|)$ time, a data structure having linear size and constant query time.[2]

[1] To see whether t is contained in $V(T_v)$ or not, it suffices to check whether the least common ancestor of t and v in T corresponds to v or not. The least common ancestor between any pair of vertices of a tree can be computed in constant time after a linear time preprocessing [13].

[2] Actually, in [9] the *bottleneck edge query* (`BEQ`) problem is considered instead. However it is easy to see that the `BEQ` and the problems `BVQ` are essentially equivalent.

In our preprocessing, we build such a structure on the tree T where each vertex $x \in T$ weighs $\ell(x)$, and then we use it to locate x in the path between v and t whenever we need to report an approximate distance for $d_{G-(u,v)}(s,t)$.

We are now ready to state the main result of this section.

▶ **Theorem 5.** *Let G be a non-negatively real weighted and undirected n-vertex and m-edge graph, and let s be a source node. There exists an EFT 2-SSDO that has size $O(n)$ and constant query time, and that can be constructed using $O(mn + n^2 \log n)$ time and $O(m)$ space.*

Proof. As we already discussed it is easy to answer a query in constant time once we store: (i) the SPT T of G w.r.t. s, (ii) the label $\ell(v)$ for each v, (iii) the value $w(\pi_{G-e}(s,v))$ for each $(u,v) \in E(T)$, and (iv) a data structure for the BVQ problem. The total space used is hence $O(n)$.

Concerning the time and the space used by Algorithm 1, observe that for each edge $e = (u,v)$, we can compute an SPT of $G - e$ with source s in $O(m + n \log n)$ time and $O(m)$ space. Therefore, for each t the distance test can be accomplished in $O(1)$ time. It remains to show that also the ancestor test can be done in constant time. To this aim, it is sufficient to maintain for each vertex x the (current) number ν_x of marked ancestors of x in T, and check whether $\nu_t - \nu_u > 0$. The maintenance of these values can be clearly done with constant time and space overhead, from which the claim follows. ◀

3 The EFT $(1+\varepsilon)$-SSDO

In this section we describe our main result, namely how to build, given any $0 < \varepsilon < 1$, an EFT $(1+\varepsilon)$-SSDO having $O\left(n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ size and $O\left(\log n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ query time.

Our distance oracle stores a set of $O\left(n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ (exact) distance values that are computed by a preprocessing algorithm that we describe below. From a high-level point of view, we follow the same approach used in the previous section, but here a vertex t can be marked several times, each corresponding to a specific failing edge $e = (u,v) \in E(T)$ for which the algorithm computes the shortest path $\pi_{G-e}(s,t)$ that is edge disjoint from $\pi_T(v,t)$. We will show that such paths have strictly decreasing lengths and that they are $O\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ in number. We will store all these distance values and we will show that they can be used to efficiently answer any distance query by suitably combining them with distances in T.

More precisely, for every $e = (u,v) \in E(T)$ and every $t \in V(T_v)$, the preprocessing algorithm computes a value $\texttt{dist}(t,e)$ that satisfies $d_{G-e}(s,t) \leq \texttt{dist}(t,e) \leq \sqrt{1+\varepsilon} \cdot d_{G-e}(s,t)$. Furthermore, each value $\texttt{dist}(t,e)$ represents the total length of a path P from s to t in $G - e$, whose structure can be either of the following two types:

type 1: $P = \pi_{G-e}(s,t)$;

type 2: P can be decomposed into $\pi_{G-e'}(s,z)$, for some e' and z such that $\texttt{dist}(z,e') = d_{G-e'}(s,z)$, and $\pi_T(z,t)$ (possibly, either $e = e'$ or $z = t$).

Since each path of type 2 can be easily derived by combining a path of type 1 with a path in T, our oracle stores only all the values $\texttt{dist}(t,e) = d_{G-e}(s,t)$ that represent paths of type 1. In the next two subsections, we will show that, for every $e \in E(T)$ and every t, our distance oracle can compute a $(\sqrt{1+\varepsilon})$-approximation of $\texttt{dist}(t,e)$ in $O\left(\log n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ time.

3.1 The preprocessing algorithm

The preprocessing algorithm (see the pseudocode of Algorithm 2) visits all the edges of T in preorder and, for each $e = (u, v) \in E(T)$, it visits all the vertices of T_v in preorder. For the rest of this section, unless stated otherwise, let $e = (u, v)$ be a fixed edge of T that is visited by the algorithm. The algorithm sets $\texttt{dist}(v, e) = d_{G-e}(s, v)$, i.e., $\texttt{dist}(v, e)$ always represents a path of type 1. When the algorithm visits t, with $t \neq v$, it first checks whether the shortest, among several paths from s to t in $G - e$ of type 2, has a total length of at most $\sqrt{1+\varepsilon} \cdot d_{G-e}(s, t)$. If this is the case, then the algorithm sets $\texttt{dist}(t, e)$ equal to the total length of such a path, otherwise it sets $\texttt{dist}(t, e) = d_{G-e}(s, t)$, i.e., $\texttt{dist}(t, e)$ represents a path of type 1. The preprocessing algorithm returns the set of all distance values that represent the paths of type 1.

For each vertex t, the algorithm stores the total length of the last path from s to t of type 1 that has computed in the variable $\texttt{last}(t)$.

Algorithm 2: Selects paths of type 1 whose lengths are stored in the oracle.

```
   // Initialization of variables
1  $S, S' = \emptyset$ for every $t \in V(G)$ do
2      $\texttt{last}(t) = \infty$

   // All the values $\texttt{dist}(t, e)$ are computed
3  for every $e = (u, v) \in E(T)$ in preorder w.r.t. $T$ do
4      $\texttt{last}(v), \texttt{dist}(v, e) = d_{G-e}(s, v)$; add $d_{G-e}(s, v)$ to $S'$        // path of type 1
5      for every $t \in V(T_v) \setminus \{v\}$ in preorder w.r.t. $T$ do
           // The length of a path from $s$ to $t$ in $G - e$ of type 2 is computed
6          $\texttt{dist}(t, e) = \min\{\texttt{last}(z) + d_T(z, t) \mid z \in A(t, e)\}$
7          if $\texttt{dist}(t, e) > \sqrt{1+\varepsilon} \cdot d_{G-e}(s, t)$ then
8              $\texttt{last}(t), \texttt{dist}(t, e) = d_{G-e}(s, t)$; add $d_{G-e}(s, t)$ to $S$   // path of type 1

9  return $S$ and $S'$.
```

For the rest of this section, unless stated otherwise, let t be a fixed vertex of T_v that is visited by the algorithm. The proof of the following proposition is trivial.

▶ **Proposition 6.** *At the end of the visit of t, $\texttt{dist}(t, e) \leq \sqrt{1+\varepsilon} \cdot d_{G-e}(s, t)$.*

The following lemma is similar to Lemma 3 and it is useful to prove that $\texttt{dist}(t, e) \geq d_{G-e}(s, t)$.

▶ **Lemma 7.** *If $d_{G-e}(s, t)$ is added to $S \cup S'$, then $\pi_{G-e}(s, t)$ and $\pi_T(v, t)$ are edge disjoint.*

Proof. The claim trivially holds when $t = v$ since $\pi_T(v, v)$ contains no edge. Therefore, we assume that $t \neq v$. We prove the claim by contradiction by showing that if $\pi_{G-e}(s, t)$ and $\pi_T(v, t)$ were not edge disjoint, then the algorithm would not add $d_{G-e}(s, t)$ to $S \cup S'$. So, we assume that $\pi_{G-e}(s, t)$ and $\pi_T(v, t)$ are not edge disjoint. Let t' be, among the vertices that are contained in both $\pi_{G-e}(s, t)$ and $\pi_T(v, t)$, the one that is closest to v w.r.t. the number of hops in $\pi_T(v, t)$. Clearly, $t' \neq t$ and $\pi_T(t', t)$ is a shortest path from t' to t in G as well as in $G - e$. Thus, by the suboptimality property of shortest paths, $d_{G-e}(s, t) = d_{G-e}(s, t') + d_{G-e}(t', t) = d_{G-e}(s, t') + d_T(t', t)$. Let $z \in A(t', e)$ be the vertex such that $\texttt{dist}(t', e) = \texttt{last}(z) + d_T(z, t')$ (possibly $z = t'$). As the algorithm visits t' before visiting t, by Proposition 6, $\texttt{dist}(t', e) \leq \sqrt{1+\varepsilon} \cdot d_{G-e}(s, t')$ at the beginning of the visit of t. Therefore

$$\begin{aligned}
\texttt{last}(z) + d_T(z,t) &= \texttt{last}(z) + d_T(z,t') + d_T(t',t) \\
&= \texttt{dist}(t', e) + d_T(t',t) \\
&\leq \sqrt{1+\varepsilon} \cdot d_{G-e}(s,t') + d_T(t',t) \\
&\leq \sqrt{1+\varepsilon} \cdot d_{G-e}(s,t).
\end{aligned}$$

As $\texttt{dist}(t,e) \leq \texttt{last}(z) + d_T(z,t)$ already before the execution of the if statement during the visit of t, the algorithm never adds $d_{G-e}(s,t)$ to $S \cup S'$. The claim follows. ◀

We now prove that $\texttt{dist}(t,e) \geq d_{G-e}(s,t)$.

▶ **Lemma 8.** *At the end of the visit of t, $\texttt{dist}(t,e) \geq d_{G-e}(s,t)$.*

Proof. The claim trivially holds if the algorithm sets $\texttt{dist}(t,e) = d_{G-e}(s,t)$. Therefore, we need to prove the claim when the condition of the if statement during the visit of t is not satisfied, i.e., $\texttt{dist}(t,e) = \texttt{last}(z) + d_T(z,t)$, for some vertex $z \in A(t,e)$ (possibly, $z = t$). Let $\texttt{last}(z) = d_{G-e'}(s,z)$, for some $e' = (u', v')$ such that z is a vertex of $T_{v'}$ (possibly $e' = e$). We divide the proof into the following two cases according to whether $e' = e$ or not.

Consider the case in which $e' = e$ and observe that e is not contained in $\pi_T(z,t)$. Therefore $\texttt{dist}(t,e) = d_{G-e'}(s,z) + d_T(z,t) = d_{G-e}(s,z) + d_{G-e}(z,t) \geq d_{G-e}(s,t)$.

Consider the case in which $e' \neq e$ and observe that e is an edge of the path $\pi_T(v', z)$. Furthermore, $\texttt{last}(z) = d_{G-e'}(s,z)$ implies that the algorithm has added $d_{G-e'}(s,z)$ to $S \cup S'$. Therefore, by Lemma 7, $\pi_{G-e'}(s,z)$ and $\pi_T(v',z)$ are edge disjoint. This implies that e is contained neither in $\pi_{G-e'}(s,z)$ nor in $\pi_T(z,t)$. Therefore, $d_{G-e}(s,t) \leq d_{G-e'}(s,z) + d_T(z,t) = \texttt{last}(z) + d_T(z,t) = \texttt{dist}(t,e)$, and the claim follows. ◀

The following proposition allows us to prove that the number of paths of type 1 computed by the algorithm is almost linear in n.

▶ **Proposition 9.** *Let $e_0, e_1, \ldots, e_k$ be all the pairwise distinct edges of T, in the order in which they are visited by the algorithm, such that $d_{G-e_i}(s,t) \in S$. Then, for every $i = 0, 1, \ldots, k$, $d_{G-e_i}(s,t) < 2/\big((\sqrt{1+\varepsilon}-1)(1+\varepsilon)^{i/2}\big) d_G(s,t)$. Furthermore, $k < 2 \cdot \frac{\log\left(2/(\sqrt{1+\varepsilon}-1)\right)}{\log(1+\varepsilon)}$.*

Proof. Let $e_0 = (u_0, v_0)$ and observe that at the end of the visit of e_0 and v_0

$$\begin{aligned}
\texttt{last}(v_0) + d_T(v_0,t) &= d_{G-e_0}(s,v_0) + d_T(v_0,t) \\
&\leq d_{G-e_0}(s,t) + d_T(t,v_0) + d_T(v_0,t) \\
&\leq d_{G-e_0}(s,t) + 2d_T(s,t) \\
&= d_{G-e_0}(s,t) + 2d_G(s,t).
\end{aligned}$$

Since $d_{G-e_0}(s,t) \in S$, $\sqrt{1+\varepsilon} \cdot d_{G-e_0}(s,t) < \texttt{last}(v_0) + d_T(v_0,t)$, and therefore

$$d_{G-e_0}(s,t) < \frac{2}{\sqrt{1+\varepsilon}-1} d_G(s,t). \tag{1}$$

Next, observe that the value $\texttt{last}(t)$ at the beginning of the visit of edge e_i, with $1 \leq i \leq k$, is equal to $d_{G-e_{i-1}}(s,t)$. Since $d_{G-e_i}(s,t) \in S$, we have that

$$\sqrt{1+\varepsilon} \cdot d_{G-e_i}(s,t) < d_{G-e_{i-1}}(s,t) \qquad \text{for every } i = 1, \ldots, k. \tag{2}$$

Thus, if, for any $i > 0$, we combine inequality (1) and all the inequalities (2) with $j \leq i$, we obtain $(1+\varepsilon)^{i/2} d_{G-e_i}(s,t) < 2/(\sqrt{1+\varepsilon}-1) d_G(s,t)$, i.e.,

$$d_{G-e_i}(s,t) < \frac{2}{(\sqrt{1+\varepsilon}-1)(1+\varepsilon)^{i/2}} d_G(s,t).$$

Moreover, using $d_G(s,t) \leq d_{G-e_k}(s,t)$ in $d_{G-e_k}(s,t) < 2/\big((\sqrt{1+\varepsilon}-1)(1+\varepsilon)^{k/2}\big) d_G(s,t)$ we obtain $(1+\varepsilon)^{k/2} < 2/(\sqrt{1+\varepsilon}-1)$, i.e.,

$$k < 2 \cdot \frac{\log\big(2/(\sqrt{1+\varepsilon}-1)\big)}{\log(1+\varepsilon)}.$$

The claim follows. ◀

Observe that $\log\big(2/(\sqrt{1+\varepsilon}-1)\big) = O(\log(1/\varepsilon))$, and that $\log(1+\varepsilon) = \Theta(\varepsilon)$. Therefore, using Proposition 9 and the fact that $|S'| = n-1$, we obtain

▶ **Corollary 10.** $|S \cup S'| = O\left(n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$.

▶ **Lemma 11.** *Algorithm 2 can be implemented to run in $O(mn + n^2 \log n)$ time and $O\left(m + n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ space.*

Proof. First we prove the time bound. Clearly, the inizialization of variables takes $O(n)$ time. Let $e = (u,v)$ be an edge that is visited by the algorithm. The algorithm computes an SPT of $G - e$ rooted at s in $O(m + n \log n)$ time. Let $t \neq v$ be the vertex that is going to be visited by the algorithm and let t' be the parent of t in T. Observe that

$$\begin{aligned} \min_{z \in A(t,e)} \big\{\texttt{last}(z) + d_T(z,t)\big\} &= \min\left\{\texttt{last}(t), \min_{z \in A(t',e)} \big\{\texttt{last}(z) + d_T(z,t)\big\}\right\} \\ &= \min\left\{\texttt{last}(t), \min_{z \in A(t',e)} \big\{\texttt{last}(z) + d_T(z,t')\big\} + w(t',t)\right\} \\ &= \min\big\{\texttt{last}(t), \texttt{dist}(t',e) + w(t',t)\big\}, \end{aligned} \tag{3}$$

Therefore, each value $\texttt{dist}(t,e)$ can be computed in constant time rather than in $O(n)$ time. Hence, the overall running time is $O(mn + n^2 \log n)$.

Concerning the space complexity, observe that, from Equation (3), the algorithm does not need to store all the values $\texttt{dist}(t,e)$ but, for each t, it is enough to remember the last computed value $\texttt{dist}(t,e)$. This can be clearly done with an array of n elements. Next, observe that, during the visit of e, the algorithm only needs the one-to-all distances in $G - e$. This implies that there is no need to keep all the $n-1$ SPT's of $G - e$, for every $e \in E(T)$, at the same time and therefore, all these SPT's can share the same $O(n)$ space. Finally, $|S \cup S'| = O\left(n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ by Corollary 10. The claim follows. ◀

3.2 The data structure

We now describe how the values in S and S' can be organized in a data structure of size $O\left(n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ so that our distance oracle can compute a $(\sqrt{1+\varepsilon})$-approximation of $\texttt{dist}(t,e)$ in $O\left(\log n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ time.

Remind that we say that $e' \prec e''$ if the preprocessing algorithm has visited e' before visiting e'', and that we also use $e' \preceq e''$ to denote that either $e' \prec e''$ or $e' = e''$. Let $k = \left\lfloor 2 \cdot \frac{\log\left(2/(\sqrt{1+\varepsilon}-1)\right)}{\log(1+\varepsilon)} \right\rfloor$ and let $a_i = \frac{2}{(\sqrt{1+\varepsilon}-1)(1+\varepsilon)^{i/2}}$. Finally, for every $i = 0, 1, \ldots, k$, let

$$S_i = \Big\{ d_{G-e'}(s,z) \in S \mid a_{i+1} \cdot d_G(s,z) \leq d_{G-e'}(s,z) < a_i \cdot d_G(s,z) \Big\}.$$

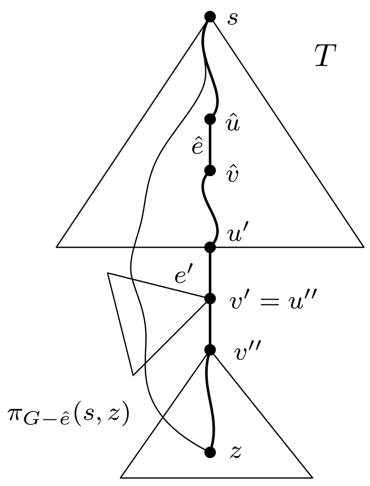

Figure 2 Representation of the proof of Proposition 12. The shortest path between s and t in T is shown. Notice that the path $\pi_{G-\hat{e}}(s,z)$ is edge disjoint from the path $\pi_T(\hat{v},z)$.

By Proposition 9, we have that $\{S_i \mid i = 0,1,\ldots,k\}$ is a partition of S.

We maintain a set of $k+1$ trees $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_k$, one for each S_i. Each tree $\mathcal{T}_i$ is a copy of T, where each vertex z, such that $d_{G-e'}(s,z) \in S_i$, has a label $\ell_i(z) = e'$. Every other vertex $z \in V(G) \setminus S_i$ has a label $\ell_i(z) = \infty$ such that $e' \prec \infty$, for every edge $e' \in E(T)$.

In the following, we denote the value of $\texttt{last}(z)$ at the end of the visit of edge e' by $\texttt{last}(z,e')$. First of all, we prove the following proposition.

▶ **Proposition 12.** *If $e' \preceq e''$, then $\texttt{last}(z,e'') \leq \texttt{last}(z,e')$.*

Proof. Let $e' = (u',v')$ and $e'' = (u'',v'')$. Notice that the claim can be proved by showing that it holds under the assumption that $v' = u''$. Furthermore, we can also assume that $\texttt{last}(z,e') \neq \infty$ as well as $\texttt{last}(z,e'') \neq \texttt{last}(z,e')$, otherwise the claim would be trivially true. This last assumption together with $v' = u''$ imply that $\texttt{last}(z,e'') = d_{G-e''}(s,z)$. Let $\texttt{last}(z,e') = d_{G-\hat{e}}(s,z)$, for some $\hat{e} \preceq e'$, with $\hat{e} = (\hat{u},\hat{v})$. Clearly, $d_{G-\hat{e}}(s,z) \in S \cup S'$. Therefore, by Lemma 7, $\pi_{G-\hat{e}}(s,z)$ and $\pi_T(\hat{v},z)$ are edge disjoint (see Figure 2). Since e'' is an edge of $\pi_T(\hat{v},z)$, $\pi_{G-\hat{e}}(s,z)$ is also a path from s to t in $G - e''$ and therefore $\texttt{last}(z,e'') = d_{G-e''}(s,z) \leq d_{G-\hat{e}}(s,z) = \texttt{last}(z,e')$. ◀

Let $e = (u,v) \in E(T)$ and let t be a vertex of T_v. Using Proposition 12, we have that either $\texttt{dist}(t,e) = \texttt{last}(v,e) + d_T(v,t) = d_{G-e}(s,v) + d_T(v,t)$, or

$$\begin{aligned}
&\texttt{dist}(t,e) = \min\left\{\texttt{last}(z,e) + d_T(z,t) \mid z \in A(t,e) \setminus \{v\}\right\} \\
&= \min\left\{\texttt{last}(z,e) + d_T(z,t) \mid z \in A(t,e)\right\} \\
&= \min_{i=0,1,\ldots,k}\left\{\min\left\{\texttt{last}(z,\ell_i(z)) + d_T(z,t) \mid z \in A(t,e) \wedge \ell_i(z) \preceq e\right\}\right\} \\
&= \min_{i=0,1,\ldots,k}\left\{\delta_i := \min\left\{d_{G-e'}(s,z) + d_T(z,t) \mid z \in A(t,e) \wedge d_{G-e'}(s,z) \in S_i \wedge e' \preceq e\right\}\right\}.
\end{aligned}$$

In the former case, $\texttt{dist}(t,e)$ is available in $O(1)$ time, since $d_{G-e}(s,v)$ is stored in S'. In the latter case, we now show how to compute, for any fixed $i = 0,1,\ldots,k$, a $(\sqrt{1+\varepsilon})$-approximate upper bound to δ_i in $O(\log n)$ time. Using Proposition 9, this will imply that our oracle is able to answer a query in $O\left(\log n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ time.

First of all, we prove that the labels of each $\mathcal{T}_i$ satisfy a nice property.

▶ **Lemma 13.** *Let z' and z'' be two distinct vertices of $A(t,e)$ such that z'' is a proper ancestor of z' and $\ell_i(z') = e'$ and $\ell_i(z'') = e''$, for some edges $e', e'' \in E(T)$, with $e', e'' \preceq e$ (possibly, $e' = e''$). We have that $d_{G-e''}(s,z'') + d_T(z'',t) \leq \sqrt{1+\varepsilon} \cdot \big(d_{G-e'}(s,z') + d_T(z',t)\big)$.*

Proof. Since $d_{G-e''}(s,z'') \in S_i$, we have that $d_{G-e''}(s,z'') < a_i \cdot d_G(s,z'')$. Furthermore, $d_{G-e'}(s,z') \in S_i$ implies that $d_{G-e'}(s,z') \geq a_{i+1} \cdot d_G(s,z') = a_i/\sqrt{1+\varepsilon} \cdot d_G(s,z')$. As a consequence, $d_{G-e''}(s,z'') + d_T(z'',t) < a_i \cdot d_G(s,z'') + d_T(z'',z') + d_T(z',t) \leq a_i \cdot d_G(s,z') + d_T(z',t) \leq \sqrt{1+\varepsilon} \cdot d_{G-e'}(s,z') + d_T(z',t) \leq \sqrt{1+\varepsilon} \cdot \big(d_{G-e'}(s,z') + d_T(z',t)\big)$. ◀

Let $z'' \in A(t,e)$ be the vertex closest to v w.r.t. T such that $\ell_i(z) = e'' \preceq e$, if such a vertex exist. Let $\delta_i = d_{G-e'}(s,z') + d_T(z',t)$, for some e' and z' such that $z' \in A(t,e)$, $d_{G-e'}(s,z') \in S_i$, and $e' \preceq e$. Observe that $d_{G-e''}(s,z'') + d_T(z'',t) \geq \delta_i$. Moreover, since z' and z'' satisfy all the hyphotesis of Lemma 13, we have that

$$\delta_i \leq d_{G-e''}(s,z'') + d_T(z'',t) \leq \sqrt{1+\varepsilon} \cdot \delta_i.$$

Therefore, the value $d_{G-e''}(s,z'') + d_T(z'',t)$ is a $(\sqrt{1+\varepsilon})$-approximate upper bound to the value δ_i. Now we show how the vertex z'' can be computed in $O(\log n)$ time.

To this aim, we preprocess each tree $\mathcal{T}_i$ in order to build a linear-size data structure that answers `BVQ` queries in constant time. This can be done in $O(n \log n)$ time per tree. We also preprocess T so we are able to perform *level-ancestor* queries in constant time. The size needed by this latter data structure is $O(n)$ and it can be built in linear-time [3, 2]. In a level ancestor query, we are given a vertex $x \in V(T)$ and a positive integer h, and we ask for the ancestor y of x such that $\pi_T(x,y)$ contains exactly h edges. We can then find z'' by performing a binary search over the vertices of $A(t,e)$, as follows.

Let $e = (u,v)$, we perform a level ancestor query on T to find the vertex x of $\pi_T(v,t)$ that divides the path into roughly two halves. Let x' be the parent of x, and let y and y' be the vertices of $\pi_T(x,t)$ and $\pi_T(v,x')$ of minimum labels, respectively. Notice that y and y' can be found in constant time by performing two `BVQ` queries on $\mathcal{T}_i$. If $\ell_i(y') \preceq e$, then we remember y' as the best vertex found so far and we iterate the binary search in $\pi_T(v,x')$. Otherwise, if $e \prec \ell_i(y')$, then we compare $\ell_i(y)$ and e. If $\ell_i(y) \preceq e$, then we remember y as the best vertex found so far and we iterate the binary search in $\pi_T(x,t)$. If $e \prec \ell_i(y)$, then we can complete our binary search and return the best vertex found, if any.

We have then proven the following:

▶ **Theorem 14.** *Let G be a non-negatively real weighted and undirected n-vertex and m-edge graph, and let s be a source node. For any arbitrarily small $0 < \varepsilon < 1$, there exists an EFT $(1+\varepsilon)$-SSDO that has size $O\left(n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ and $O\left(\log n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ query time, and that can be constructed using $O(mn + n^2 \log n)$ time and $O\left(m + n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ space.*

4 Lower bounds on the size of additive EFT ASPT and SSDO

In this section, we give a lower bound on the bit size of an EFT $\beta(d)$-additive SSDO. Recall that after the failure of any edge, such an oracle must return an estimation d' of the actual distance d between s and any node such that $d \leq d' \leq d + \beta(d)$, where β is any positive real function. Due to space limitations, the proof of next theorem is omitted and will be given in the full version of the paper.

▶ **Theorem 15.** *Let $\beta(d) = kd^{1-\delta}$, for arbitrary $k \geq 1$ and $0 < \delta \leq 1$. Then, there exist classes of polynomially weighted graphs with n nodes such that:*

1. *any* EFT *$\beta(d)$-additive ASPT has $\Omega(n^2)$ edges;*
2. *any* EFT *$\beta(d)$-additive SSDO has $\Omega(n^2)$ bit size for at least an input graph, regardless of its query time.*

References

1 Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 66(1):18–50, 2013. doi:10.1007/s00453-012-9621-y.

2 Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004. doi:10.1016/j.tcs.2003.05.002.

3 Omer Berkman and Uzi Vishkin. Finding level-ancestors in trees. *J. Comput. Syst. Sci.*, 48(2):214–230, 1994. doi:10.1016/S0022-0000(05)80002-9.

4 Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *STOC*, pages 101–110, 2009.

5 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-tolerant approximate shortest-path trees. In *ESA*, pages 137–148, 2014. doi:10.1007/978-3-662-44777-2_12.

6 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *STACS*, pages 18:1–18:14, 2016.

7 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-sensitivity distance oracles and routing schemes. In *ESA*, pages 84–96, 2010.

8 Annalisa D'Andrea, Mattia D'Emidio, Daniele Frigioni, Stefano Leucci, and Guido Proietti. Path-fault-tolerant approximate shortest-path trees. In *SIROCCO*, pages 224–238, 2015. doi:10.1007/978-3-319-25258-2_16.

9 Erik D. Demaine, Gad M. Landau, and Oren Weimann. On cartesian trees and range minimum queries. *Algorithmica*, 68(3):610–625, 2014. doi:10.1007/s00453-012-9683-x.

10 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. doi:10.1137/S0097539705429847.

11 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *SODA*, pages 506–515, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496826.

12 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *FOCS*, pages 748–757, 2012.

13 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. doi:10.1137/0213024.

14 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.

15 Merav Parter. Dual failure resilient BFS structure. In *PODC*, pages 481–490, 2015.

16 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *ESA*, pages 779–790, 2013. doi:10.1007/978-3-642-40450-4_66.

17 Merav Parter and David Peleg. Fault tolerant approximate BFS structures. In *SODA*, pages 1073–1092, 2014.

18 Christian Sommer. Shortest-path queries in static networks. *ACM Comput. Surv.*, 46(4):45:1–45:31, 2014. doi:10.1145/2530531.

19 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. A preliminary version of this paper appeared in *SODA'01*. doi:10.1145/1044731.1044732.