Generic Single Edge Fault Tolerant Exact Distance Oracle

Manoj Gupta
IIT Gandhinagar, India
gmanoj@iitgn.ac.in

Aditi Singh
IIT Gandhinagar, India
aditi.singh@iitgn.ac.in

April 28, 2018

Abstract

Given an undirected unweighted graph G and a source set S of $|S| = \sigma$ sources, we want to build a data structure which can process the following query $\mathrm{Q}(s, t, e)$: find the shortest distance from s to t avoiding an edge e, where $s \in S$ and $t \in V$. When $\sigma = n$, Demetrescu, Thorup, Chowdhury and Ramachandran (SIAM Journal of Computing, 2008) designed an algorithm with $\tilde{O}(n^2)$ space[1] and $O(1)$ query time. A natural open question is to generalize this result to any number of sources. Recently, Bilò et. al. (STACS 2018) designed a data-structure of size $\tilde{O}(\sigma^{1/2}n^{3/2})$ with the query time of $O(\sqrt{n\sigma})$ for the above problem. We improve their result by designing a data-structure of size $\tilde{O}(\sigma^{1/2}n^{3/2})$ that can answer queries in $\tilde{O}(1)$ time.

In a related problem of finding fault tolerant subgraph, Parter and Peleg (ESA 2013) showed that if detours of *replacement* paths ending at a vertex t are disjoint, then the number of such paths is $O(\sqrt{n\sigma})$. This eventually gives a bound of $O(n\sqrt{n\sigma}) = O(\sigma^{1/2}n^{3/2})$ for their problem. *Disjointness of detours* is a very crucial property used in the above result. We show a similar result for a subset of replacement path which **may not** be disjoint. This result is the crux of our paper and may be of independent interest.

[1] $\tilde{O}(\cdot)$ hides poly $\log n$ factor.

1 Introduction

Real life graph networks like communication network or road network are prone to link or node failure. Thus, algorithms developed for these networks must be resilient to failure. For example, the shortest path between two nodes may change drastically even if a single link fails. So, if the problem forces us to find shortest paths in the graph, then it should find the next best shortest path after a link failure. There are many ways to model this process: one of them is *fault-tolerant graph algorithm*. In this model, we have to preprocess a graph G and build a data-structure that can compute a property of the graph after any k edges/vertices of the graph have failed. Note the difference between this model and *dynamic graph model*. In a dynamic graph algorithm, we have to *maintain* a property of a continuously changing graph. However, in the fault tolerant model, we expect the failure to be repaired readily and restore our original graph.

In this paper, we study the shortest path problem in the fault tolerant model. Formally, we are given an undirected and unweighted graph G and a source set S of $|S| = \sigma$ sources. We want to build a data structure which can process the following query $\mathrm{Q}(s, t, e)$: find the shortest distance from s to t avoiding an edge e, where $s \in S$ and $t \in V$. Such a data-structure is also called a *distance oracle*. When there are n sources, Demetrescu et al. [9] designed an oracle that can find the shortest path between any two vertices in G after a single vertex/edge failure in $\tilde{O}(n^2)$ space and $O(1)$ query time. Recently, Bilò et. al. [5] generalized this result to any number of sources. They designed a data-structure of size $\tilde{O}(\sigma^{1/2} n^{3/2})$ with the query time of $O(\sqrt{n\sigma})$ for the above problem.

To understand our problem, we should also understand a closely related problem of finding *fault tolerant subgraph*. Here, we have to find a subgraph of G such that BFS tree from $s \in S$ is preserved in the subgraph after any edge deletion. In an unweighted graph, a BFS tree preserves the shortest path from s to all vertices in G. Parter and Peleg [17] showed that a subgraph of size $O(\sigma^{1/2} n^{3/2})$ is both necessary and sufficient to solve the above problem. The above result indicates that there should be a better fault-tolerant distance oracle for any value of σ.

Inspired by this result, we generalize the result of [9] to any number of sources – by showing that there exists a distance oracle of size $\tilde{O}(\sigma^{1/2} n^{3/2})$ which can answer queries in $\tilde{O}(1)$ time. Note that our result nearly matches the space bound achieved by Parter and Peleg[17] – up to $\operatorname{polylog} n$ factors. We now state the main result of this paper formally:

Theorem 1. *There exists a data-structure of size $\tilde{O}(\sigma^{1/2} n^{3/2})$ for multiple source single fault tolerant exact distance oracle that can answer each query in $\tilde{O}(1)$ time.*

This generalization turns out to be much more complex than the result in [9]. Indeed, the techniques used by Demetrescu et al. [9] are also used by us to weed out *easy replacement* paths. To take care of other paths, we take an approach similar to Parter and Peleg[17]. They used the following trick: if the *detour* of replacement paths are *disjoint*, then the number of such paths can be bounded easily by a *counting argument*. The main challenge is then to show that paths in question are indeed disjoint – which is also easy in their problem. We use a technique similar to above – however, our paths are not disjoint, they may intersect. We believe that this technique can be of independent interest and may be used in solving closely related fault tolerant subgraph problems.

1.1 Related Work

Prior to our work, the work related to fault tolerant distance oracle was limited to two special cases, $\sigma = 1$ or $\sigma = n$. As stated previously, Demetrescu et al. [9] designed a single fault tolerant distance oracle of size $\tilde{O}(n^2)$ with a query time of $O(1)$. The time to build the data-structure is

$O(mn^2)$ – which was improved to $O(mn \log n)$ by Bernstein and Karger [4]. The above result also works for a directed weighted graph. Pettie and Duan [10] were able to extend this result to two vertex faults. The size and query time of their distance oracle is $\tilde{O}(n^2)$ and $\tilde{O}(1)$ respectively. If the graph is weighted, then Demetrescu et al. [9] showed that there exists a graph in which a single vertex fault tolerant distance oracle will take $\Omega(m)$ space. Recently, Bilò et. al. [5] designed the following data-structure: for every $S, T \subseteq V$, a data-structure of size $\tilde{O}(n\sqrt{|S||T|})$ and query time $O(\sqrt{|S||T|})$, where the query asks for the shortest distance from $s \in S$ to $t \in T$ avoiding any edge. If $|S| = \sigma$ and $|T| = n$, then the size of their data-structure is $\tilde{O}(\sigma^{1/2}n^{3/2})$ and the query time is $O(\sqrt{n\sigma})$.

The next set of results are not *exact* but *approximate*, that is, they return an approximate distance (by a multiplicative *stretch* factor) between two vertices after an edge/vertex fault. Also, these oracles work for a single source only. Baswana and Khanna [14] showed that a 3-stretch single source single fault tolerant distance oracle of size $\tilde{O}(n)$ can be built in $\tilde{O}(m + n)$ time and a constant query time. Bilò et. al.[6] improved the above result: a distance oracle with stretch 2 of size $O(n)$ and $O(1)$ query time. In another result, Bilò et. al. [7] designed a k fault tolerant distance oracle of size $\tilde{O}(kn)$ with a stretch factor of $(2k + 1)$ that can answer queries in $\tilde{O}(k^2)$ time. The time required to construct this data-structure is $O(kn\alpha(m, n))$, where $\alpha(m, n)$ is the inverse of the Ackermann's function. If the graph is unweighted, then Baswana and Khanna[14] showed that a $(1 + \epsilon)$-stretch single source fault tolerant distance oracle of size $\tilde{O}(\frac{n}{\epsilon^3})$ can be built in $O(m\sqrt{n/\epsilon})$ time and a constant query time. Bilò et. al [6] extended this result for weighted graph by designing a distance oracle with stretch $(1 + \epsilon)$ of size $O(\frac{n}{\epsilon} \log \frac{1}{\epsilon})$ and a logarithmic query time.

There is another line of work, called the *replacement path* problem. In this problem, we are given a source s and destination t and for each edge e on the shortest st path, we need to find shortest s to t path avoiding e. The problem can be generalized to finding k shortest s to t path avoiding e. The main goal of this problem is to find all shortest paths as fast as possible. Malik et al. [15] showed that in an undirected graphs, replacement paths can be computed in $O(m + n \log n)$ time. For directed, unweighted graphs, Roditty and Zwick [19] designed an algorithm that finds all replacement paths in $O(m\sqrt{n})$ time. For the k-shortest paths problem, Roditty [18] presented an algorithm with an approximation ratio 3/2, and the running time $O(k(m\sqrt{n} + n^{3/2} \log n))$. Bernstein [3] improved the above result to get an approximation factor of $(1 + \epsilon)$ and running time $O(km/\epsilon)$. The same paper also gives an improved algorithm for the approximate st replacement path algorithm. See also [11, 22, 21].

As mentioned previously, a problem closely related to our problem is the fault tolerant subgraph problem. The aim of this problem is to find a subgraph of G such that BFS tree from $s \in S$ is preserved in the subgraph after any edge deletion. Parter and Peleg [17] designed an algorithm to compute single fault tolerant BFS tree with $O(n^{3/2})$ space. They also showed their result can be easily extended to multiple source with $O(\sigma^{1/2}n^{3/2})$ space. Moreover, their upper bounds were complemented by matching lower bounds for both their results. This result was later extended to dual fault BFS tree by Parter [16] with $O(n^{5/3})$ space. Gupta and Khan [12] extended the above result to multiple sources with $O(\sigma^{1/3}n^{5/3})$ space. All the above results are optimal due to a result by Parter [16] which states that a multiple source k fault tolerant BFS structure requires $\Omega(\sigma^{\frac{1}{k+1}}n^{2-\frac{1}{k+1}})$ space. Very recently, Bodwin et. al. [8] showed the existence of a k fault tolerant BFS structure of size $\tilde{O}(k\sigma^{1/2^k}n^{2-1/2^k})$.

Other related problems include fault-tolerant DFS and fault tolerant reachability. Baswana et al. [1] designed an $\tilde{O}(m)$ sized fault tolerant data structure that reports the DFS tree of an undirected graph after k faults in $\tilde{O}(nk)$ time. For single source reachability, Baswana et al. [2] designed an algorithm that finds a fault tolerant reachability subgraph for k faults using $O(2^k n)$

edges.

1.2 Comparison with Previous Technique

Our technique should be directly compared to the technique in Bilò et. al.[5]. We discuss their work when $|S| = 1$ and $|T| = n$, that is we want a single source fault tolerant distance oracle. Consider any st path where $t \in V$. For the last $\sqrt{n}\log n$ edges in this st path (edges from the vertex t), we explicitly store the shortest replacement paths avoiding these edges. For the remaining replacement paths, note that the length of these replacement paths is always $\geq \sqrt{n}\log n$. So, we sample a set R of $O(\sqrt{n})$ vertices. With very high probability, one of our sampled vertices will lie on the replacement path (Lemma 1, [5]). We can also show that the shortest path from the sampled vertex to t will never contain the edge avoided by these remaining replacement paths (Lemma 3, [5]). We can store all shortest paths from $v \in R$ to $t \in V$ using space $O(n^{3/2})$.

Thus, we have reduced the problem of finding a replacement paths from s to vertices in V, to finding replacement path from s to vertices in R. This reduction is useful as there are just $O(\sqrt{n})$ vertices in R. Fortunately, there already exists a data-structure [9], say D, that can solve the reduced problem in $O(n^{3/2})$ space and $O(1)$ query time.

Now the query algorithm in [5] is straightforward. Consider the query $Q(s,t,e)$. If e is one of the last $\sqrt{n}\log n$ edges on st path, then we have already stored the replacement path. Else, we know that the replacement path avoiding e must pass through a vertex in R. Unfortunately, we don't know that vertex. So, we try out all the vertices in R. That is, for each $v \in R$, we find the shortest path from s to v avoiding e (using data-structure D in $O(1)$ time) and add it to shortest vt distance. Thus, we have to return the minimum of all the computed shortest paths. This gives us the running time of $|R| = O(\sqrt{n})$.

To improve upon the techniques in [5], we use the following strategy: we also sample R vertices of size $\tilde{O}(\sqrt{n})$. Instead of looking at all the vertices in R, we concentrate on the vertex of R that lies on the st path, say v. If the replacement path passes through v, then we can find it in $O(1)$ time using D (as done in [5]). The main novel idea of this paper is to show that the number of replacement paths that do not pass through v is $O(\sqrt{n})$. This helps us in reducing the running time from $O(\sqrt{n})$ to $\tilde{O}(1)$. Moreover, we show that this technique can be generalized to any number of sources.

2 Preliminaries

We use the following notation throughout the paper:

- xy : Given two vertices x and y, let xy denote a path between x and y. Normally this path will be the shortest path from x to y in G. However, in some places in the paper, the use of xy will be clear from the context.
- $|xy|$: It denotes the number of edges in the path xy.
- $(\cdot \diamond \cdot)$: Given two paths sx and xt, $sx \diamond xt$ denotes the concatenation of paths sx and xt.
- *after or below/before or above* x : We will assume that the st path (for $s \in S$ and $t \in V$) is drawn from top to bottom. Assume that $x \in st$. The term *after or below* x on st path refers to the path xt. Similarly *before or above* x on st path refers to the path sx.
- *replacement path*: The shortest path that avoids any given edge is called a *replacement* path.

3 Our Approach

We will randomly select a set of terminals $\mathcal{T}$ by sampling each vertex with probability $\sqrt{\frac{\sigma}{n}}$. Note that the size of $\mathcal{T}$ is $\tilde{O}(\sqrt{\sigma n})$ with high probability. For a source s and $t \in V$, let t_s be the last terminal encountered on the st path. The following lemma is immediate:

Lemma 2. *If $|st| \geq c\sqrt{\frac{n}{\sigma}}\log n$ $(c \geq 3)$, then $|t_s t| = \tilde{O}(\sqrt{\frac{n}{\sigma}})$ with a very high probability for all $s \in S$ and $t \in V$.*

Proof. Let $E_{s,t}$ be the event that none of the last $c\sqrt{\frac{n}{\sigma}}\log n$ vertices on st path are in $\mathcal{T}$. So, $\mathsf{P}[E_{st} \text{ occors}] = (1 - \sqrt{\frac{\sigma}{n}})^{c\sqrt{\frac{n}{\sigma}}\log n} \leq \frac{1}{n^c}$. Using union bound, $\mathsf{P}[\cup_{s,t} E_{s,t} \text{ occors}] \leq n\sigma\frac{1}{n^c} \leq \frac{1}{n^{c-2}}$. Thus, with a very high probability $|t_s t| = \tilde{O}(\sqrt{\frac{n}{\sigma}})$ for all $s \in S$ and $t \in V$.

□

Let G_p denote the graph where each edge is perturbed by a weight function that ensures unique shortest paths. Our st path is the shortest s to t path in G_p, let us denote its length by $|st|_p$. Note that G_p contains a unique shortest path between any two vertices, even the ones that avoid an edge – such a graph has been used before in related problems [4, 17, 13]. We can use G_p even to find all the replacement paths. However, we want our replacement paths to have other nice property, that is, *the length replacement paths(without perturbation) from s to t are different.* This property is not satisfied by replacement paths in G_p. We employ another simple strategy to find a replacement path. Following [12], we define preferred replacement paths:

Definition 3. *A path P is called a **preferred** replacement path from s to t avoiding e if (1) it diverges and merges the st path* just once *(2) it divergence point from the st path is as close to s as possible (3) it is the shortest path in G_p satisfying (1) and (2).*

The replacement path has to diverge from the st path before e. Ideally, we want a replacement path that diverges from st path as close to s as possible. This is a crucial feature which will ensure that all replacement paths from s to t have different lengths. The first condition ensures that we do not diverge from st path just to get a higher point of divergence. If many shortest paths are diverging from a same vertex, the third condition is used to break ties. In the ensuing discussion, we will assume that we are always working with a preferred replacement path.

The initial st path is found out by finding the unique shortest path in G_p. Consider the query $Q(s,t,e)$. If the failed edge e does not lie on st path, then we can report $|st|$ as the shortest distance from s to t avoiding e. To this end, we should be able to check whether e lies in the shortest path from s to t. At this point, we will use the property of graph G_p. If $e(u,v)$ lies in st path, then we have to check if u and v lie on st path. To this end, we check if $|su|_p + |ut|_p = |st|_p$ and $|sv|_p + |vt|_p = |st|_p$. If both the above two equations are satisfied then the st path passes through e (as the shortest path from u to v is 1). We can also find whether u or v is closer to s on st path. Without loss of generality assume that u is closer to s than v on st path.

However, we do not have space to store all these distances. Specifically, the second term on the LHS of above two equations mandates that we store the distance of every pair of vertices in the graph. This implies that the size of our data structure is $O(n^2)$ which is not desirable.

To solve the above problem, we observe that if e lies in the $t_s t$ path, then we have just enough space to store this fact. So, given any e, we can easily find if $e \in t_s t$. If $e \in st_s$, then we know that $|su|_p + |ut_s|_p + |t_s t|_p = |st|_p$ and $|sv|_p + |vt_s|_p + |t_s t|_p = |st|_p$. This equality is easier to check with the space at hand. So, we have the following two cases:

1. (Near Case) e lies on $t_s t$.
2. (Far Case) e lies on st_s.

3.1 Handling the Near Case

For each $e(u, v) \in t_s t$, let P_e be the preferred replacement path from s to t avoiding e. We put $(e, |P_e|)$ in a balanced binary search tree $\textsc{Bst}(s, t)$ with the key being e. Given any query $Q(s, t, e)$, we now need to check if e lies in $\textsc{Bst}(s, t)$. This can be done in $\tilde{O}(1)$ time and the length of the preferred replacement path can be reported.

The space required for $\textsc{Bst}(s, t)$ is directly proportional to the size of path $t_s t$. By Lemma 2, we know that $|t_s t| = \tilde{O}(\sqrt{\frac{n}{\sigma}})$. Thus, the size of $\textsc{Bst}(s, t) = \tilde{O}(\sqrt{\frac{n}{\sigma}})$. This implies that the cumulative size of all the associated binary search tree is $\cup_{t \in V} \cup_{s \in S} |t_s t| = \tilde{O}(n\sigma\sqrt{\frac{n}{\sigma}}) = \tilde{O}(\sigma^{1/2} n^{3/2})$.

3.2 Handling the Far Case

We first need to check if $e \in st_s$. To this end we use the following data-structures.

- B_0: For each pair of vertices x and y where $x \in (S \cup \mathcal{T})$ and $y \in V$, the shortest path between x and y in G and G_p is stored in $B_0(x, y)$ and $B_0^p(x, y)$ respectively. The total size of B_0 is $\tilde{O}((\sigma + \sqrt{n\sigma})n) = \tilde{O}(\sigma^{1/2} n^{3/2})$.
- B_1: For each pair of vertices $s \in S$ and $t \in V$, $B_1(s, t)$ contains the vertex in $\mathcal{T}$ closest to t on st path, that is t_s. The total size of B_1 is $O(\sigma n) = \tilde{O}(\sigma^{1/2} n^{3/2})$.

To check if $e(u, v) \in st_s$, we first find $t_s \leftarrow B_1(s, t)$. Then we check if $B_0^p(s, u) + B_0^p(u, t_s) + B_0^p(t_s, t) = B_0^p(s, t)$ and $B_0^p(s, v) + B_0^p(v, t_s) + B_0^p(t_s, t) = B_0^p(s, t)$. If yes, then $e \in st_s$. We subdivide the far case into two more sub-cases:

1. The preferred replacement path avoiding e passes through t_s.
2. The preferred replacement path avoiding e avoids t_s.

The first case turns out to be a generalization of techniques used by Demetrescu et. al.[9] to solve the all pair distance oracle under single edge/vertex failure – we will use the compact version of this algorithm presented by Pettie and Duan [10]. The second case is a *new and unexplored* case. We will show that we can bound the number of preferred replacement paths in this case to $O(\sqrt{n\sigma})$ for a fixed vertex t. This would imply that the total number of such paths is $O(\sigma^{1/2} n^{3/2})$. We are able to bound the number of paths even though these paths may intersect with each other – this is a new feature of our analysis which is much different from the analysis done by Parter and Peleg [17] on a related problem.

Section 4 deals with the first case. In Section 5, we will apply our new approach to the special case when $\sigma = 1$, or there is a single source. In Section 6, we will discuss the potential problems in extending our approach to multiple sources. Section 7 and 8 extends our approach to multiple sources and in Section 9 we develop our data-structure that can answer queries in $\tilde{O}(1)$ time.

4 Preferred replacement path passes through t_s

Under this assumption, we only need to find the preferred replacement path from s to t_s avoiding e. Note that we already know $|t_s t|$ length via $B_0(t_s, t)$. We use the following additional data structures:

- B_2: For $x \in S \cup \mathcal{T}$ and $y \in V$, $B_2(x, y)$ contains the vertex w on xy path such that $|yw| = 2^{\lfloor \log |xy| \rfloor}$, or in words, the vertex nearest to x whose distance from y is a power of 2. The size of B_2 is $\tilde{O}((\sigma + \sqrt{n\sigma})n) = \tilde{O}(\sigma^{1/2} n^{3/2})$.
- B_3: For $x \in V$ and $y \in \mathcal{T}$, $B_4(x, y, \oplus 2^i)$ contains the shortest path from x to y avoiding every 2^i-th edge on the path xy from x (where $i \leq \log\lfloor |xy| \rfloor$). Since $|\mathcal{T}| = \tilde{O}(\sqrt{n\sigma})$, the size of B_3 is $\tilde{O}(\sigma^{1/2} n^{3/2})$.
- B_4: For $s \in S$ and $x \in V$, $B_4(s, x, \ominus 2^i)$ contains the shortest path from s to x avoiding the 2^i-th edge on the path sx from x (where $i \leq \log\lfloor |sx| \rfloor$). The size of B_4 is $\tilde{O}(n\sigma) = \tilde{O}(\sigma^{1/2} n^{3/2})$.

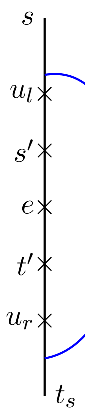

Figure 1: The hardest part of the distance oracle, when the replacement path neither passes through u_l not u_r.

- B_5: For every $s \in S$ and $x \in \mathcal{T}$, $B_5(s, x, [\oplus 2^i, \ominus 2^j])$ contains the shortest path from s to x avoiding the sub path that start from 2^i-th vertex from s and ends at 2^j-th vertex from x on the path sx (where $i, j \leq \log\lfloor sx \rfloor$). The size of B_5 is $\tilde{O}(\sigma^{3/2} n^{1/2}) = \tilde{O}(\sigma^{1/2} n^{3/2})$.

Assume that we get a query Q$(s, t, e(u, v))$, that is, find the shortest path from s to t avoiding e. Assume without loss of generality that u is closer to s on st path than v. We answer this query as follows: first we find the distance of v from s, via $B_0(s, v)$. If $B_0(s, v)$ is a power of 2 then we can directly use $B_3(s, t_s, \oplus|sv|) + B_0(t_s, t)$. Else, let $u_l \leftarrow B_2(s, v)$ and $u_r \leftarrow B_2(t_s, u)$. Here u_l is the nearest vertex to s on vs path whose distance from v is a power of 2. Similarly, u_r is the nearest vertex to t_s on ut_s path whose distance form u is a power of 2. There are three cases now:

1. The preferred replacement path passes through u_l.
2. The preferred replacement path passes through u_r.
3. The preferred replacement path neither passes through u_l nor u_r.

For the first case, we can report $B_0(s, u_l) + B_3(u_l, t_s, \oplus|u_l v|) + B_0(t_s, t)$. For the second case, we can report $B_4(s, u_r, \ominus|uu_r|) + B_0(u_r, t_s) + B_0(t_s, t)$. The hardest case is when the preferred replacement path neither passes through u_l nor u_r. Let s' be the farthest vertex on sv path from s whose distance is a power of 2, that is $|ss'| = 2^{\lfloor \log |sv| \rfloor}$. Similarly let t' be the farthest vertex on $t_s u$ path from t_s whose distance is a power of 2. Now, we can report the shortest path from s to t_s avoiding $[s', t']$ – this is also stored in $B_5(s, t_s, [\oplus 2^{\lfloor \log |sv| \rfloor}, \ominus 2^{\lfloor \log |t_s u| \rfloor}])$. By construction, if the shortest path avoids $[u_l, u_r]$, then it also avoids $[s', t']$. Thus we can return $B_5(s, t_s, [\oplus 2^{\lfloor \log |sv| \rfloor}, \ominus 2^{\lfloor \log |t_s u| \rfloor}]) +\ B_0(t_s, t)$ as the length of the preferred replacement path. The reader can check that the running time of our algorithm is $O(1)$ as we have to check $B_0, B_1, B_2, B_3, B_4, B_5$ constant number of times. For the correctness of the above procedure, please refer to [9, 10].

Now, we move on to the harder case, that is, replacement paths avoid t_s too. For this, we will fix a vertex t. We will show that the query $Q(s, t, e(u, v))$ can be answered in $\tilde{O}(1)$ using $\tilde{O}(\sqrt{\sigma n})$ space. This immediately implies that we can answer exact queries in $\tilde{O}(1)$ time using $\tilde{O}(\sigma^{1/2} n^{3/2})$ space.

5 Preferred Replacement path avoids t_s

Handling preferred replacement paths that avoid t_s turns out to be a challenging and unexplored case. For better exposition, we will first solve the problem for the case when $\sigma = 1$, that is there is only one source. Let $\mathcal{R}$ be the set of all preferred replacement paths from s to t that do not pass through t_s. We make two important observations:

1. The size of $\mathcal{R}$ is $O(\sqrt{n})$.
2. Preferred replacement paths in $\mathcal{R}$ avoid one contiguous sub path of st.

Few remarks are in order. If the preferred replacement paths in $\mathcal{R}$ were disjoint, then bounding the size of $\mathcal{R}$ is easy. However, we are able to bound the size of $\mathcal{R}$ even if paths are intersecting. The second observation implies that we can build a balanced binary search tree containing paths in $\mathcal{R}$. Each node in this tree will contain a preferred replacement path P. The key for each node will be the start and end vertex of the sub path P avoids. We will use this BST to find an appropriate replacement path that avoids an edge e.

Definition 4. *(Detour of a replacement path) Let P be a preferred replacement path avoiding an edge e on st path. Then detour of P is defined as,* $\textsc{Detour}(P) := P \setminus st$*. That is, detour is a path the leaves st before e till the point it merges back to st again.*

Since our replacement path P also avoids t_s, the following lemma is immediate by the definition of preferred path.

Lemma 5. *Let P be a preferred replacement path in $\mathcal{R}$ that avoids e and t_s on st path, then (1)* $\textsc{Detour}(P)$ *cannot merge back to st_s path and (2)* $\textsc{Detour}(P)$ *is a contiguous path.*

Lemma 6. *Let $P, P' \in \mathcal{R}$ avoid e and e' respectively on st_s path. Also assume that e is closer to s than e'. Then (1) P avoids e' (2)* $\textsc{Detour}(P')$ *starts after e on st_s path and (3) $|P| > |P'|$.*

Proof. (1) P diverges from st_s path above e. Since $P \in \mathcal{R}$, it merges back on t_st path only. Since $e, e' \in st_s$ (we are in *far* case) and e is closer to s than e', this implies that P also avoids e'.
(2) Assume that $\textsc{Detour}(P')$ starts above e on st_s path. This implies that both P and P' avoid e and e'. But then, our algorithm will choose one of these two paths as a preferred path that avoids both e and e'. Thus, we arrive at a contradiction as there are two different preferred replacement paths avoiding e and e'.
(3) Since P avoids e' (by (1)) and $\textsc{Detour}(P')$ starts after e (by (2)), P' is the preferred path to avoid e' only if $|P'| < |P|$ (else P would be the preferred path as it leaves the st path earlier than P').
□

The converse of the third part of the lemma is also true. Since we will be using it in future, we prove it now.

Lemma 7. *Let P and P' be two preferred replacement paths that avoid e and e' on st path respectively. If $|P| > |P'|$, then e is closer to s than e'.*

Proof. Assume for contradiction that e' is closer to s than e. Since the replacement path P' has to diverge from st before e' and merge again only in t_st, P' also avoid e. But then P' should be the replacement path for avoiding e too, as $|P'| < |P|$, a contradiction. □

By Lemma 6(3), we know that all preferred replacement paths in $\mathcal{R}$ have different lengths. In fact, it is the main reason we defined a preferred replacement path. We can thus arrange these paths in decreasing order of their lengths. Thus, we get the following corollary.

Corollary 8. *Given a set $\mathcal{R}$ of preferred replacement paths from s to t (that also avoid t_s), we can arrange paths in decreasing order of their lengths.*

Given a path $P \in \mathcal{R}$, let $(< P)$ be the set of all preferred replacement paths with length less than P. Similarly, let $(> P)$ be the set of all preferred replacement paths with length greater than P. If P avoids e, then by Lemma 7, it also avoids all edges avoided by paths in $(< P)$. By Lemma 6, for any path $P' \in (< P)$, $\textsc{Detour}(P')$ starts after e on st_s path. We will now show a simple but important property of a path P in $\mathcal{R}$.

Lemma 9. *Let $P \in \mathcal{R}$ be the shortest path from s to t avoiding e such that $|P| = |st| + \ell$ where $\ell \geq 0$, then the size of the set $(< P)$ is $\leq \ell$.*

Proof. Since a path in $(< P)$ avoids some edge in st path, its length has to be $\geq |st|$. By Corollary 8, all paths in $\mathcal{R}$, and thus $(< P)$ have different lengths. But the length of paths in $(< P)$ is less than the length of P. Thus, there can be atmost ℓ paths in $(< P)$. □

Definition 10. *(Unique path of P) Let $\textsc{Unique}(P)$ be the prefix of $\textsc{Detour}(P)$ which does not intersect with any detours in $\cup_{P' \in (>P)} \textsc{Detour}(P')$.*

We now arrange all preferred replacement paths in $\mathcal{R}$ in decreasing order of their lengths. Assume that we are processing a path P according to this ordering such that P avoids e on st path. If $|\textsc{Unique}(P)| \geq \sqrt{n}$, then we have associated $O(\sqrt{n})$ vertices on $\textsc{Unique}(P)$ to P. Else $\textsc{Unique}(P) < \sqrt{n}$ and we have the following two cases:

5.1 Detour(P) does not intersect with detour of any path in $(> P)$

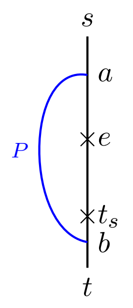

Figure 2: $\textsc{Detour}(P)$ does not intersect detour of any path in $(> P)$

Let $\textsc{Detour}(P)$ start at a and end at b – the vertex where it touches t_st path. Let ab denote the path from a to b on P. By our assumption $\textsc{Unique}(P) = ab$ and $|ab| < \sqrt{n}$. By Lemma 6, all replacement paths in $(< P)$ pass through e (as detour of these replacement paths start below e) and by Lemma 7, these replacement paths avoid edges that are closer to t than e. We can view the replacement paths as if they are starting from the vertex a. That is, consider paths $\{P \setminus sa\} \cup \{P' \setminus sa|\ P' \in (< P)\}$. These replacement paths avoid edges in at. $|P \setminus sa| = |ab| + |bt| \leq |ab| + |at| < |at| + \sqrt{n}$. Applying Lemma 9, we infer that the number of paths in $\{P' \setminus sa | P' \in (< P)\}$ is $\leq \sqrt{n}$

5.2 Detour(P) intersects with detour of a path in $(> P)$

Assume that P first intersects with $P' \in (> P)$. Let P' avoid e' and $\textsc{Detour}(P')$ start at a' and end at b' (see Figure 3). Let us assume that $\textsc{Detour}(P)$ starts at a and it intersects $\textsc{Detour}(P')$ at c. This implies that $\textsc{Unique}(P) = ac$.

Consider the path $sa' \diamond a'c \diamond ca \diamond at$. We claim that this path avoids e'. This is due to the fact that by Lemma 6, Detour(P) starts after e' on st path. So, ca and at avoids e'. Since $P' = sa' \diamond a'c \diamond cb' \diamond b't$, length of P' must be $\leq$ length of the alternate path. Thus,

$$\begin{array}{llll} & |sa'| + |a'c| + |cb'| + |b't| & \leq & |sa'| + |a'c| + |ca| + |at| \\ \implies & |cb'| + |b't| & \leq & |ca| + |at| \\ \implies & |ac| + |cb'| + |b't| & \leq & 2|ca| + |at| \end{array}$$

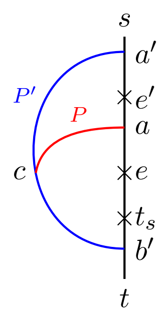

Figure 3: Detour(P) intersects first with Detour(P') at c where $P' \in (> P)$.

On the left hand of the inequality, we have a path from a to t avoiding e. So, its length should be $\geq$ length of the preferred path $P \setminus sa$. Thus $|P \setminus sa| \leq 2|ca| + |at| \leq 2\sqrt{n} + |at|$. By Lemma 6, all replacement paths in $(< P)$ pass through e (as detour of these replacement paths start below e) and by Lemma 7, these replacement paths avoid edges that are closer to t than e. We can view the replacement paths as if they are starting from the vertex a. That is, consider paths $\{P \setminus sa\} \cup \{P' \setminus sa | P' \in (< P)\}$. Applying lemma 9, we infer that the number of paths in $\{P' \setminus sa | P' \in (< P)\}$ is $\leq 2\sqrt{n}$.

Our arguments above point to the following important observation: *Once we find a replacement path in $\mathcal{R}$ with unique path length $< \sqrt{n}$, then there are at most $2\sqrt{n}$ replacement paths in $\mathcal{R}$ left to process.* Since there can be at most $\sqrt{n}$ paths in $\mathcal{R}$ with unique path length $\geq \sqrt{n}$, we have proven the following lemma:

Lemma 11. $|\mathcal{R}| = O(\sqrt{n})$.

We now build a data-structure which will exploit Lemma 11. However, we need another key but simple observation. By Lemma 6, if $|P| > |P'|$, then Detour(P') starts below the edge avoided by P. This lemma implies that Detour(P') starts below all edges avoided by P. Thus P avoids some contiguous path in st_s and detour of all replacement paths in $(< P)$ start below the last edge (which is closer to t_s) in this subpath. Thus, we have proved the second key lemma:

Lemma 12. *A replacement path P avoids a contiguous subpath of st.*

Let First(P) and Last(P) denote the first and the last vertex of the contiguous path that P avoids. Given a vertex v, let $v.depth$ denote the depth of v in the BFS tree of s. We can store the depth of all vertices in an array (takes $O(n)$ space). Lastly, we build a balanced binary search tree BST(t) in which each node represents a path P. The key used to search the node is the range: [First(P).$depth$, Last(P).$depth$]. By Lemma 12, all replacement paths avoid contiguous subpaths of st_s. These contiguous paths are also disjoint as there is only one preferred path avoiding an edge. Thus, the key we have chosen forms a total ordered set with respect to the relation $\{<, >\}$. The size of BST(t) is $O(\sqrt{n})$ as the size of $\mathcal{R}$ is $O(\sqrt{n})$. We are now ready to process any query Q($s, t, e(u, v)$). We just need to search for an interval in BST(t) that contains $u.depth$ and $v.depth$. This can be done in $\tilde{O}(1)$ time. Thus we have proved the following theorem:

Theorem 13. *There exists a data-structure of size $\tilde{O}(n^{3/2})$ for single source single fault tolerant exact distance oracle that can answer each query in $\tilde{O}(1)$ time.*

6 From Single Source to Multiple Sources

Unfortunately, the analysis for the single source case is not easily extendible to multiple source case. We identify the exact problem here. Consider the case described in Section 5.2. In this case, we show that if $|P'| > |P|$ and P intersects with P', then there is a path available for P (that is $ac \diamond cb' \diamond b't$). We can use this path because it also avoids e (the edge avoided by path P). First, we show that the above assertion is not true when we move to multiple source case. Consider the following example (See Figure 4). Here, P avoids e on st path and P' avoids e' on $s't$ path. DETOUR(P) starts at a and its intersects P' at c. DETOUR(P') starts at a' and it hits st path at b' and then passes through e. Note that the full path P from s to t is not shown in Figure 4. The reader can check that the path $ac \diamond cb' \diamond b't$ is not an alternate path for P as it passes through e. We say that such a path is a bad path because it breaks the easy analysis of single source case (we will formally define bad paths in Section 8.2). However, we are able to show that the total number of *good* paths (paths which are not bad) is $\geq$ the number of bad paths. Good paths exhibit properties similar to the set $\mathcal{R}$ in Section 5. This will help us in bounding them (and thus bad paths too).

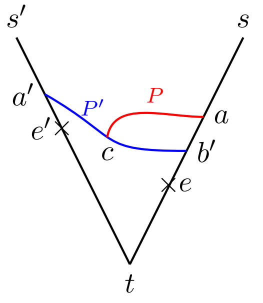

Figure 4: The bad case for us: $P' \in (> P)$ intersects with P and then passes through the edge e that P avoids.

Once again we will fix a vertex t and show that the number of replacement paths from $s \in S$ to t that also avoids t_s is $O(\sqrt{\sigma n})$. Let BFS(t) denote the union of all shortest paths from t to $s \in S$. The reader can check that the union of these paths does not admit a cycle, so we can assume that its a tree rooted at t. Since BFS(t) has at most σ leaves, the number of vertices with degree > 2 in BFS(t) is $O(\sigma)$. We now contract all the vertices of degree 2 (except t and $s \in S$) in BFS(t) to get a tree that only contains leaves of BFS(t), the root t, all the sources and all other vertices with degree > 2 in BFS(t).

Definition 14. *(σ-BFS(t)) σ-BFS(t) is a tree obtained by contracting all the vertices with degree exactly 2 in BFS(t) except t and source $s \in S$.*

Definition 15. *(Intersection vertex and segment in σ-BFS(t))*
Each node σ-BFS(t) is called an intersection vertex. An edge $xy \in \sigma$-BFS(t) denotes a path between two vertices in BFS(t). We call such an edge in σ-BFS a segment. We use this term in order to differentiate between edges in BFS(t) and σ-BFS(t). Also, we will use the following convention: if xy is a segment, then y is closer to t than x.

σ-BFS(t) has at most σ vertices with degree ≤ 2. This implies that there are at most $O(\sigma)$ intersection vertices and segments in σ-BFS(t).

As in the single source case, we would like to find the preferred path for each avoided edge on the st path where $s \in S$. However, we don't have enough space to store all these paths. Also storing all paths seems wasteful. Consider two preferred replacement paths P and P' that start from s and s' respectively. These two paths meet at an intersection vertex x after which they are same, that is, they take the same detour to reach t. Storing both P and P' seems wasteful as they are essentially the same path once they hit x. To this end, we only store preferred path corresponding to each segment in σ-BFS(t). We now describe our approach in detail.

Let xy be a segment in σ-BFS(t). We divide replacement paths whose detour start in xy into two types:

$\mathcal{R}_1(xy)$: Preferred replacement paths from x to t whose detour starts in xy but the avoided edge lies in yt_x.

$\mathcal{R}_2(xy)$: Preferred replacement paths from x to t whose start of detour and avoided edge both lie strictly inside segment xy (that is, detour cannot start from x or y).

Let $\mathcal{R}_1 := \cup_{xy \in \sigma\text{-BFS}(t)} \mathcal{R}_1(xy)$ and $\mathcal{R}_2 := \cup_{xy \in \sigma\text{-BFS}(t)} \mathcal{R}_2(xy)$. The set $\mathcal{R}_1$ helps us to weed out simple preferred replacement paths. We will show that we can store preferred replacement paths in $\mathcal{R}_1$ in $O(\sigma)$ space – one per segment in σ-BFS(t). The hardest case for us in $\mathcal{R}_2$, which contains bad paths. Let $\mathcal{B}$ denote the set of bad paths in $\mathcal{R}_2$. We will show that $|\mathcal{B}| \leq |\mathcal{R}_2 \setminus \mathcal{B}|$ (the number of bad paths is $\leq$ number of good paths in $\mathcal{R}_2$) and $|\mathcal{R}_2 \setminus \mathcal{B}| = O(\sqrt{n\sigma})$ (the number of good path is $O(\sqrt{n\sigma})$). This implies that $|\mathcal{R}_2| = O(\sqrt{n\sigma})$.

Since $\mathcal{R}_1$ and $\mathcal{R}_2$ are of size $O(\sqrt{n\sigma})$, we can make a data-structure of size $O(\sqrt{n\sigma})$. In this data-structure, we have stored a preferred path for each segment. However, we have to answer queries of type $\mathrm{Q}(s,t,e)$ where s is a source. In Section 9, we will see how to use preferred paths of segments to answer queries in $\tilde{O}(1)$ time.

7 Analysing preferred replacement paths in $\mathcal{R}_1$

We first show the following:

Lemma 16. *For each segment $xy \in \sigma$-BFS(t), $|\mathcal{R}_1(xy)| = 1$*

Proof. Assume that there are two preferred paths P and P' whose detour start in xy and their avoided edge e and e'(respectively) lie in yt_x. Since we are analyzing paths in the *far case*, detours of P and P' meet xt path in t_xt subpath. Since both e and e' lie on yt_x path, this implies that both P and P' avoid e and e'. Thus, we will choose the smaller of the two paths as the preferred path avoiding both e and e'. Else if $|P| = |P'|$, then we will choose that path which leaves the xt path as early as possible. Thus, there is only one preferred path avoiding both e and e', a contradiction. □

The above lemma implies that $|\mathcal{R}_1| = \cup_{xy \in \sigma\text{-BFS}(t)} |\mathcal{R}_1(xy)| = O(\sigma) = O(\sqrt{n\sigma})$.

8 Analysing preferred replacement paths in $\mathcal{R}_2$

We first show that one special kind of path will never lie in $\mathcal{R}_2$. This characterization will help in analyzing bad paths in $\mathcal{R}_2$.

Lemma 17. *Let P be a preferred path from x to t avoiding e on xt path. If P merges with any segment $x'y'$ and then diverges from $x't$ path, then $P \notin \mathcal{R}_2$.*

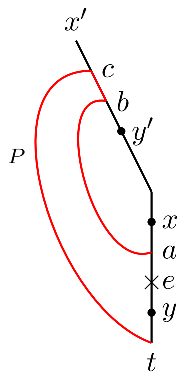

Figure 5: The path P merges with another segment $x'y'$ and then diverges.

Proof. Once P merges with $x'y'$ segment, it will diverge from it only if $x't$ contains e. This implies that xt and $x't$ intersect or xt is a subpath of $x't$.
Assume for contradiction that $P \in \mathcal{R}_2$, that is, DETOUR(P) start strictly inside segment xy. Consider the Figure 5 in which the detour of the preferred replacement path P starts after x (at a). It then intersect $x'y'$ at b and then diverges from $x't$ path at c. We claim that there exists another path to reach b that is shorter than $xa \diamond ab$. This path is xb, where path xb is a subpath of $x't$. It is also a shorter path (in G_p) as it uses edges in original $x't$ path.

This path (1) diverges at x and (2) is shorter (in G_p) as it uses edges from $x't$ path. Thus, there is a shorter replacement path than P that avoids e. This path is not in $\mathcal{R}_2$ as its detour starts at x . This leads to a contradiction as we had assumed that P is the preferred replacement path avoiding e. Thus our assumption, namely that $P \in \mathcal{R}_2$ must be false. $\square$

We will now analyze paths in $\mathcal{R}_2$. Consider two replacement paths P, P' avoiding edges e, e' (respectively) on $xy, x'y'$ segment respectively. Let a, a' be the starting vertex of DETOUR(P), DETOUR(P') respectively. We say that $P \prec P'$ if $|at| < |a't|$. If $|at| = |a't|$, then the tie is broken arbitrarily.
Given a path $P \in \mathcal{R}_2$, let $(< P)$ be the set of all replacement paths in $\mathcal{R}_2$ that are $\prec P$ in the ordering. Similarly, $(> P)$ is the set of all replacement paths $P' \in \mathcal{R}_2$ for which $P \prec P'$. Define UNIQUE(P) according to this ordering (see definition 10). Assume that we are processing a replacement path P according to this ordering. If $|\text{UNIQUE}(P)| \geq \sqrt{n/\sigma}$, then we can associate $O(\sqrt{n/\sigma})$ *unique* vertices to P. Otherwise $|\text{UNIQUE}(P)| < \sqrt{n/\sigma}$ and we have the following two cases:

8.1 Detour(P) does not intersect with any other detour in $(> P)$

This case is similar to the first case in Section 5.1. Assume that P avoids an edge e on segment xy. Let DETOUR(P) start at $a \in xy$ and end at b – the vertex where it touches $t_x t$ path. Let ab denote the path from a to b on P. By our assumption UNIQUE$(P) = ab$ and $|ab| < \sqrt{n/\sigma}$. Consider the following set of replacement paths $(< P)_x := \{P' \in (< P) \mid P'$ avoids an edge on xy segment$\}$. By Lemma 6, all replacement paths in $(< P)_x$ pass through e (as detour of these replacement paths start below e) and by Lemma 7, these replacement paths avoid edges that are closer to y than e. We can view these replacement paths as if they are starting from vertex a. $|P \setminus xa| = |ab| + |bt| \leq \sqrt{n/\sigma} + |at|$. Using lemma 9, total number of paths in $(\leq P)_x$ is $\leq \sqrt{n/\sigma}$. Thus, once we get a replacement path $P \in \mathcal{R}_2(xy)$ with $|\text{UNIQUE}(P)| < \sqrt{n/\sigma}$, then there are at most $\sqrt{n/\sigma}$ replacement paths in $\mathcal{R}_2(xy)$ remaining to be processed. Thus, total number of paths in $\mathcal{R}_2$ with $|\text{UNIQUE}(P)| < \sqrt{n/\sigma}$ is $\sum_{xy \in \sigma\text{-BFS}(t)} \sqrt{n/\sigma} = O(\sqrt{n\sigma})$ (as there are $O(\sigma)$ segments in σ-BFS(t))

8.2 Detour(P) intersects with detour of a path in $(> P)$

We first give a formal definition of a bad path that was defined informally in Section 6.

Definition 18. *(Bad Path) A path $P \in \mathcal{R}_2$ is called a bad path if there exists another path $P' \in (> P)$ such that (1) $\textsc{Detour}(P)$ intersects with $\textsc{Detour}(P')$ and (2) $\textsc{Detour}(P')$ passes through the edge avoided by P after their intersection. We also say that P is a bad replacement path due to P' if P' satisfies the above two conditions.*

A path that is not bad is called a good path. In Section 6, we saw that bad paths break the easy analysis of the single source case. So, we have two cases depending on whether the path is good or bad. Let us look at the easier case first.

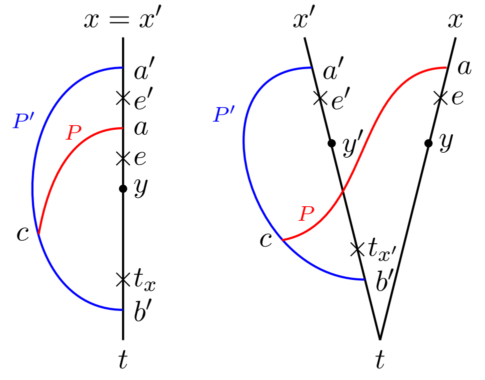

Figure 6: The figure shows two representative examples when ca and at does not pass through e'.

(1) *P is a good path.*
Assume that $P \in \mathcal{R}_2(xy)$ and it avoids an edge $e \in xy$. Assume that P intersects first with $P' \in (> P)$ and P' avoids e' on $x'y'$ segment. Note that x may be equal to x'. Let $\textsc{Detour}(P')$ start at a' and end at b'. Assume that $\textsc{Detour}(P)$ starts at a and it intersects $\textsc{Detour}(P')$ at c. Consider the path $x'a' \diamond a'c \diamond ca \diamond at$. Since $x'a' \diamond a'c$ is a part of P', it avoids e'. However, it is not clear whether $ca \diamond at$ avoids e' too. In Figure 6, we see two representative examples in which ca and at avoid e'. We will now show that both ca and at cannot passes through e'.

(a) Assume that ca passes through e'

If $x = x'$, then by Lemma 6, as $P' \in (> P)$, $\textsc{Detour}(P)$ (and hence ca) starts below e'. Thus, ca cannot pass through e' as $\textsc{Detour}(P)$ does not intersect xt_x path and $e' \in xt_x$. So let us assume that $x \neq x'$. This implies that P intersects $x'y'$. After intersecting $x'y'$ path P did not follow $x'y' \diamond y't$ (since ca intersect $\textsc{Detour}(P')$ at c). This implies that P intersect with another path $x'y'$ and then diverges. By Lemma 17, $P \notin \mathcal{R}_2$. This leads to a contradiction as we assumed that $P \in \mathcal{R}_2$. Thus our assumption, namely ca passes through e' is false.

(b) Assume that at passes through e'

If $x = x'$, then by Lemma 6, starting vertex of $\textsc{Detour}(P)$, that is a, starts below e'. Thus, at cannot pass through e'. So let us assume that $x \neq x'$. If at passes through e', then segment $x'y'$ is a subpath of xt path. This is due to the fact that $a \in xy$ and $e' \in x'y'$. Since $P' \in \mathcal{R}_2(x'y')$, $\textsc{Detour}(P')$ starts strictly inside segment $x'y'$, at vertex a'. This implies that $|a't| < |at|$. This contradicts our assumption that $P' \in (> P)$. Thus, at cannot pass through e'.

Thus, the path $x'a' \diamond a'c \diamond ca \diamond at$ is indeed a valid replacement path from x' to t avoiding e'. Since $P' = x'a' \diamond a'c \diamond cb' \diamond b't$, length of P' must be $\leq$ length of this alternate path. Thus,

$$\begin{array}{lll} & |x'a'| + |a'c| + |cb'| + |b't| & \leq & |x'a'| + |a'c| + |ca| + |at| \\ \implies & |cb'| + |b't| & \leq & |ca| + |at| \\ \implies & |ac| + |cb'| + |b't| & \leq & 2|ca| + |at| \end{array}$$

On the left hand of the inequality, we have a path from a to t avoiding e (since we know that P is a good path, so P' and thus $cb' \diamond b't$ does not pass through e). So, its length should be $\geq$ length of the preferred path $P \setminus xa$. Thus $|P \setminus xa| \leq 2|ca| + |at| \leq 2\sqrt{n/\sigma} + |at|$ (since $|\text{UNIQUE}(P)| = |ac| < \sqrt{n/\sigma}$). Consider the following set of replacement paths $(< P)_x := \{P' \in (< P) \mid P' \text{ avoids an edge on } xy \text{ segment}\}$. By Lemma 6, all replacement paths in $(< P)_x$ pass through e (as detour of these replacement paths start below e) and by Lemma 7, these replacement paths avoid edges that are closer to y than e. Applying Lemma 9, we get that the number of replacement paths $(< P)_x$ is $\leq 2\sqrt{n/\sigma}$. Thus, once we get a replacement path $P \in \mathcal{R}_2(xy)$ with $|\text{UNIQUE}(P)| < \sqrt{n/\sigma}$, then there are at most $2\sqrt{n/\sigma}$ replacement paths in $\mathcal{R}_2(xy)$ remaining to be processed. Thus, total number of paths $\in \mathcal{R}_2$ with $|\text{UNIQUE}(P)| < \sqrt{n/\sigma}$ is $\sum_{xy \in \sigma\text{-BFS}(t)} 2\sqrt{n/\sigma} = O(\sqrt{n\sigma})$ (as there are $O(\sigma)$ segments in σ-BFS(t)).

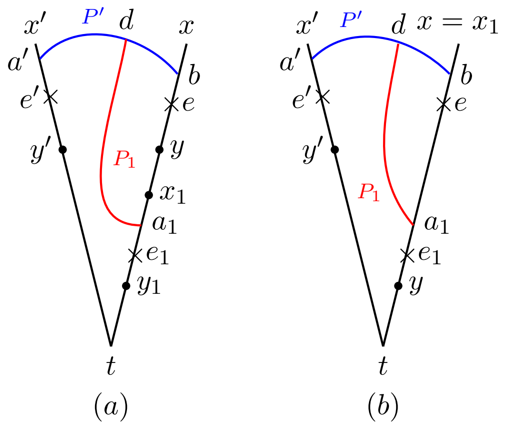

Figure 7: Two cases in which P' passes through e_1 after intersecting with P_1

(2) *P is a bad path.*
We now arrive at our hardest scenario. We will first show that the number of good paths in $\mathcal{R}_2$ is greater than the number of bad paths in $\mathcal{R}_2$. To this end, we will prove the following lemma:

Lemma 19. *For each $P' \in \mathcal{R}_2$, there exists only one replacement path $P \in \mathcal{R}_2$ which is bad due to P'.*

Proof. Assume that P is the preferred path from x to t avoiding e on segment xy. Similarly, P' is the preferred path from x' to t avoiding e' on segment $x'y'$ and P is bad due to P'. Assume for contradiction that there is one more replacement path P_1 which is bad due to P'. Let us assume that P_1 avoids e_1 on x_1y_1 segment. If $x' = x$ or $x' = x_1$, then DETOUR(P') cannot pass through e or e_1 respectively as DETOUR(P') starts before e or e_1 (as $P' \in (> P)$ or $P' \in (> P_1)$) and touches $x't$ path only at $t_{x'}t$. So, let us assume that $x' \neq x$ and $x' \neq x_1$.

Since $P' \in \mathcal{R}_2$, by lemma 17, we know that it follows xt path after hitting segment xy, say at b. This implies that e_1 also lies on the xt path. Thus, xt and x_1t intersect. Without loss of generality, assume that the segment x_1y_1 is a subpath of xt. Let us assume that DETOUR(P_1) starts at a_1 and it hits P' at d. There are two ways for P_1 to reach d (both avoiding e_1): $x_1a_1 \diamond a_1d$ and $x_1b \diamond bd$ where the first path is using the detour of P_1 and the second path uses xt path to reach b.

Note that the second path leaves the x_1t path earlier than the first path (x_1 compared to a_1 if $x \neq x_1$ (Figure 7(a)) and b compared to a_1 if $x = x_1$ (Figure 7(a))). Even then the preferred path used the first alternative. This implies that the length of the first path must be *strictly* less than the second. Thus, $|x_1a_1| + |a_1d| < |x_1b| + |bd|$. Thus,

$$|a_1d| < |db| + |bx_1| - |x_1a_1| \tag{1}$$

P' takes the following path $x'a' \diamond a'b \diamond bt$. But there is an alternative path available for P', it is $x'a' \diamond a'd \diamond da_1 \diamond a_1t$. The path is a valid path avoiding e' only if a_1d does not pass through e'. All other components of this path are a part of P' ($a'd \in a'b$ and $a_1t \in bt$) .

If a_1d does not pass through e', then we can show that the alternative path has a length less than $|P'|$, thus arriving at a contradiction. Consider the length of the alternative path:

$$\begin{aligned}&|x'a'|+|a'd| + |da_1| + |a_1t| \\ &\quad < |x'a'| + |a'd| + |db| + |bx_1| - |x_1a_1| + |a_1t| \qquad \text{(Using Equation (1))}\end{aligned}$$

If $x \neq x'$(See Figure 7(a)), then $|bx_1| - |x_1a_1| + |a_1t| \leq |bx_1| + |x_1a_1| + |a_1t| = |bt|$. Else if $x = x'$ (See Figure 7(b)), then $|bx_1| - |x_1a_1| + |a_1t| = -|ba_1| + \ |a_1t| \leq |ba_1| + \ |a_1t| = |bt|$

$$\begin{aligned}&\leq |x'a'| + |a'd| + |db| + |bt| \\ &= |x'a'| + |a'b| + |bt| \\ &= |P'|\end{aligned}$$

This leads to a contradiction as we have assumed that P' is the shortest path from x' to t avoiding e'.

To end this proof, we will show that a_1d cannot pass through e'. Assume for contradiction that a_1d passes through e'. This implies that DETOUR(P_1) intersects with $x'y'$ segment (as $e' \in x'y'$) and then diverges from it (as DETOUR(P_1) intersect with DETOUR(P') at d). By Lemma 17, $P_1 \notin \mathcal{R}_2$. But this cannot be the case as we have assumed that $P_1 \in \mathcal{R}_2$. Thus our assumption, namely a_1d passes through e' must be false.

□

The above lemma can be used to discard bad paths from $\mathcal{R}_2$. For each such discarded path, there exists at least one good path. And by the above lemma, each such good path can be used to discard at most one bad path. Thus the number of good paths in $\mathcal{R}_2$ is $\geq$ number of bad paths in $\mathcal{R}_2$. We have already shown that the total number of good paths in $\mathcal{R}_2$ is $O(\sqrt{n\sigma})$. Thus the total number of paths in $\mathcal{R}_2$ is also $O(\sqrt{n\sigma})$.

9 Building the Data Structure

Let us first recognize a potential problem in using $\mathcal{R}_1(\cdot)$. Let s_1t and s_2t path meet at vertex x (See Figure 8). Another path s_3t meets s_2t path at y where y is closer to t. $\mathcal{R}_1(s_2x)$ is the shortest path from s_1 to t avoiding e and $\mathcal{R}_1(xy)$ is the shortest path from x to t avoiding $e \in yt$. This immediately leads to the following problem. Assume that the query is $\mathrm{Q}(s_2, t, e)$ and the preferred path avoiding e is in $\mathcal{R}_1$. Then there are two candidate paths that avoid e: one that goes from s_2 to the intersection vertex x and then take path $\mathcal{R}_1(xy)$ and the other $\mathcal{R}_1(s_2x)$. Thus, we need to check these two paths and return the minimum of the two. One can make a bigger example in which there are σ segments between s_2 and t and thus we have to check $O(\sigma)$ path before we can answer the query. The problem appears because we don't know from which segment the shortest path avoiding e started its detour. If this information is not there, then it seems that we have to look at all the segments between s_2 ans t. To end this dilemma, we use heavy light decomposition of σ-BFS(t)

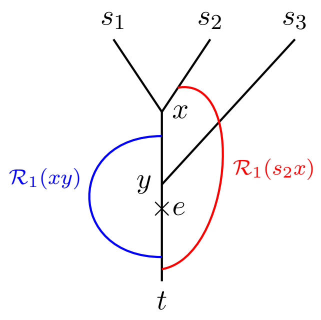

Figure 8: The shortest path from s_2 to t avoiding e can be $\mathcal{R}_1(xy)$ or $\mathcal{R}_1(s_2x)$.

[20]. For any segment $xy \in \sigma$-BFS(t) (by our convention y is closer to t), x is a *heavy* child of y if the number of nodes in the subtree under x is $\geq 1/2$(number of nodes in the subtree under y) else it is called a *light child* (or light *segment* in our case). It follows that each intersection vertex has exactly one heavy child and each vertex is adjacent to atmost two heavy edges. A *heavy chain* is a concatenation of heavy edges. A *heavy subpath* is a subpath of a heavy chain. The following lemma notes a well known property of heavy-light decomposition.

Lemma 20. *The path from a source s to t in σ-BFS(t) can be decomposed into $O(\log n)$ heavy subpaths and light segments .*

Given any source $s \in S$, by Lemma 20, the path from t to s may contain many heavy subpaths. Let $C(pq)$ be a heavy chain that starts at p and ends at q (where q is closer to t than p). A ts path may follow a heavy chain $C(pq)$ but may exit this chain from a vertex midway, say at r. Let $(C(pq), r)$ be a tuple associated with s such that the shortest path from t to s enters this heavy chain via q and leaves this chain at r. We keep a list $\textsc{Heavy}(s,t)$ which contains all the tuples $(C(pq), r)$ sorted according to the distance of heavy chain from t (that is distance qt). By Lemma 20, the size of $\textsc{Heavy}(s,t) = O(\log n)$. Similarly, we have one more list to store the light segments. $\textsc{Light}(s,t)$ contains all the light segments on the st path again ordered according to their distance from t in σ-BFS(t). Again by Lemma 20, the size of $\textsc{Light}(s,t) = O(\log n)$. Note that the size of these additional two data-structures is $\sum_{s \in S} O(\log n) = \tilde{O}(\sigma) = \tilde{O}(\sqrt{n\sigma})$.

Our main problem was that we have to find the minimum $\mathcal{R}_1(\cdot)$ of $O(\sigma)$ segments if there is a path of length σ between s and t. The trick we use here is that finding minimum on any heavy subpath takes $\tilde{O}(1)$ time. Since there are $O(\log n)$ heavy subpaths, the total time taken to find the minimum on heavy subpaths in $\tilde{O}(1)$. Also, since the number of light segments is also $O(\log n)$ finding the minimum among these also takes $\tilde{O}(1)$ time.

We now describe our intuition in detail. Let xy be a segment in a heavy chain $C(pq)$. We want to represent $\mathcal{R}_1(xy)$ in a balanced binary search tree $\textsc{Bst}(C)$. To this end, we will add a node with the tuple $(x.depth, |px \diamond \mathcal{R}_1(xy)|, |px|)$ in $\textsc{Bst}(C)$. The first element in this tuple is the depth of x in BFS(t) – it also acts as the key in this binary search tree. The second element is the path $\mathcal{R}_1(xy)$ concatenated with px. This concatenation is done so that all paths in $\textsc{Bst}(C)$ start from p and comparing two paths in $\textsc{Bst}(C)$ is possible. The third element will be used to get the path length $\mathcal{R}_1(xy)$ (by subtracting it from the second element) when need arises. Now we can augment this tree so that the following range minimum query can be answered in $\tilde{O}(1)$ time: $\textsc{Rmq}(C(pq),[a,b])$: Find minimum of $\{|px \diamond \mathcal{R}_1(xy)| \mid xy$ is a segment in heavy chain $C(pq)$ and $x.depth \geq a.depth$ and $x.depth \leq b.depth\}$. The size of $\cup_{C\in\textsc{Heavy}(s,t)}\textsc{Bst}(C)$ is $O(\sigma) = O(\sqrt{n\sigma})$ as there are at most $O(\sigma)$ segments in σ-BFS(t).

Algorithm 1: Finding the shortest replacement path (in $\mathcal{R}_1$) from s to t avoiding $e(u,v)$

```
1  Let $x$ be the first intersection point on $us$ path;
2  $min \leftarrow \infty$ ;
3  do
4      if $x$ lies in a heavy chain then
5          Using $\textsc{Heavy}(s,t)$, find $(C(pq),r)$, that is $r$ is the vertex from which $us$ path leaves
            the chain $C$;
6          $min \leftarrow \min\{min, \textsc{Rmq}(C,[x,r]) - |pr| + |sr|\}$;
7          $x \leftarrow r$;
8      end
9      else if $x$ is an endpoint of a light segment then
10         Let $x'x$ be the light segment ending at $x$ in $\sigma$-BFS$(t)$ ;   // Can be found out via
            $\textsc{Light}(s,t)$ in $\tilde{O}(1)$ time.
11         $min \leftarrow \min\{min, |\mathcal{R}_1(x'x)| + |sx'|\}$;
12         $x \leftarrow x'$;
13     end
14 while $x$ is not equal to $s$;
```

Given any edge $e(u,v)$ on st path, we can now find the shortest path in $\mathcal{R}_1$ from s to t avoiding e (see Algorithm 1). We first find the first intersection vertex on the us path from u. Let this vertex be x. We will see that finding x is also not a trivial problem – we will say more about this problem later. Now, we will go over all possible replacement paths from u to s. Thus, we search if there exists any heavy chain in $\textsc{Heavy}(s,t)$ that contains x. To this end, we first check if x lies in some light segment (this can be checked in $\tilde{O}(1)$ time). If not, then x lies in some heavy chain. We now search each heavy chain in $\textsc{Heavy}(s,t)$ to find a node x' with the smallest depth such that $x'.depth > x.depth$. Let this node be x'. Thus we have found the segment $x'x$ where x is closer to t than x'. We can easily calculate $x.depth$ as $|st| - |sx|$ or $B_0(s,t) - B_0(s,x)$. Since there are $\tilde{O}(1)$ heavy chain in $\textsc{Heavy}(s,t)$, the time taken to find if $x'x$ exists in some heavy chain is $\tilde{O}(1)$.

Assume that we found out that $x'x \in C(pq)$, and ts path leaves the chain C at r, then we want to find the shortest replacement path from r to t avoiding e. This can be found out via the range minimum query $\textsc{Rmq}(C(p,q),[x,r])$. However, note that each replacement path in C starts from p. So, we need to remove $|pr|$ from the replacement path length returned by $\textsc{Rmq}$ query. The length pr can be found out in the node $r \in \textsc{Bst}(C)$. Finally, we add $|sr|$ to get the path from s to t.

Similarly, we can process a light segment in $O(1)$ time (please refer to Algorithm 1). Thus, the time taken by Algorithm 1 is $\tilde{O}(1)$ as the while loop runs at most $O(\log n)$ times and each step in the while loop runs in $\tilde{O}(1)$ time.

9.1 Answering queries in $\tilde{O}(1)$ time

Given a query $Q(s,t,e(u,v))$, we process it as follows (assuming that e lies on st_s path (that is the *far case)* and v is closer to t than u)

1. Find the first intersection vertex on us path.
   As we have mentioned before, this is not a trivial problem. Let the first intersection vertex from u to s in $\textsc{Bst}(t)$ be denoted by $\textsc{Int}_s(u,t)$. We will first show that $\textsc{Int}_s(u,t)$ is independent of s.

   Lemma 21. *$\textsc{Int}_s(u,t) = \textsc{Int}_{s'}(u,t)$ for $s, s' \in S$ for all $u \in \mathrm{BFS}(t)$.*

   Proof. We will prove this by induction on the nodes of $\mathrm{BFS}(t)$ from leaf to root t. The base case is a leaf in $\mathrm{BFS}(t)$, that is a source vertex, which by definition, itself is an intersection vertex. For any node u, if u is an intersection vertex, $\textsc{Int}_s(u,t) = \textsc{Int}_{s'}(u,t) = u$. Else, u is a node of degree 2 in $\mathrm{BFS}(t)$. Assume that u' is the child of u in $\mathrm{BFS}(t)$. So, $\textsc{Int}_s(u,t) = \textsc{Int}_s(u',t)$ and $\textsc{Int}_{s'}(u,t) = \textsc{Int}_{s'}(u',t)$. But by induction hypothesis, $\textsc{Int}_s(u',t) = \textsc{Int}_{s'}(u',t)$. □

   We will drop the subscript s from the definition of $\textsc{Int}(\cdot,\cdot)$ as we now know that it is independent of s. We use the above lemma to construct two data structures that will help us in finding $\textsc{Int}(u,t)$.

   - $I_1(t)$: For any $u \in V$, if $\textsc{Int}(u,t)$ is within a distance of $c\sqrt{n/\sigma}\log n$ (for some constant c) from u, then we store the tuple $(u, \textsc{Int}(u,t))$ in the balanced binary search tree $I_1(t)$. For any intersection vertex $x \in \sigma$-$\mathrm{BFS}(t)$, we store at most $\tilde{O}(\sqrt{n/\sigma})$ tuples in $I_1(t)$. For a fixed t, the total space taken by $I_1(t)$ is $\tilde{O}(\sigma.\sqrt{(n/\sigma)}) = \tilde{O}(\sqrt{n\sigma})$ (as there are $O(\sigma)$ intersection vertices in σ-$\mathrm{BFS}(t)$.
   - $I_2(t)$: If u is not present in $I_1(t)$, then $\textsc{Int}(u,t)$ is at a distance $\geq c\sqrt{n/\sigma}\log n$ from u. We now use a different strategy to find $\textsc{Int}(u,t)$. We first find $u_s \leftarrow B_1(s,u)$, that is the vertex in $\mathcal{T}$ closest to u in su path. With a high probability, u_s is closer to u than $\textsc{Int}(u,t)$ and all the vertices from u to u_s have degree exactly 2 in $\mathrm{BFS}(t)$. Thus, $\textsc{Int}(u_s,t) = \textsc{Int}(u,t)$ – we will now use this property (a similar property was also used in the proof of Lemma 21). For each $x \in \mathcal{T}$ such that x is not an intersection vertex in $\mathrm{BFS}(t)$, we store the tuple $(x, \textsc{Int}(x,t))$ in another balanced binary search tree $I_2(t)$. For a fixed vertex t, the size of this data-structure is $\tilde{O}(\sqrt{n\sigma})$ space as there is only one intersection vertex for each vertex in $\mathcal{T}$ and $|\mathcal{T}| = \tilde{O}(\sqrt{n\sigma})$

   If $\textsc{Int}(u,t)$ is indeed at a distance $\leq c\sqrt{n/\sigma}\log n$, then we can use $I_1(t)$ to find it in $\tilde{O}(1)$ time, else we use $I_2(t)$ to find $\textsc{Int}(u_s,t)$ in $\tilde{O}(1)$ time.

2. Find the replacement path avoiding u if it lies in $\mathcal{R}_1$.
   To this end, we use our Algorithm 1. The first non-trivial part of this algorithm, that is, finding the first intersection vertex on the us path has already been tackled in the point above. So we can find such a replacement path (if it exists) in $\tilde{O}(1)$ time and $\tilde{O}(\sqrt{n\sigma})$ space.

3. Find the replacement path avoiding $e(u,v)$ if it lies in $\mathcal{R}_2$.
   This part is similar to our data-structure in single source case. Let $x \leftarrow \textsc{Int}(u,t)$. Using $\textsc{Heavy}(s,t)$ and $\textsc{Light}(s,t)$, in $\tilde{O}(1)$ time, we can find the segment $xy \in \sigma$-$\mathrm{BFS}(t)$ such that y is closer to t than x. In this case, we want to check if there exists any replacement path that starts in the same segment in which e resides. This replacement path first takes sx path and then takes the detour strictly inside the segment xy. All such paths are stored in $\mathcal{R}_2(xy)$ with the contiguous range of edges that they avoid on xy. We now just need to check if u and v lie in the range of some replacement path. To this end, we find $u.depth \leftarrow |st| - |su| = B_0(s,t) - B_0(s,u)$ and $v.depth \leftarrow |st| - |sv| = B_0(s,t) - B_0(s,v)$. Now we check if $u.depth$ and $v.depth$ lie in contiguous range of some replacement path in $\mathcal{R}_2(xy)$. If yes, then we return the length of that

path concatenated with sx. Note that we have already stored $|sx|$ in $B_0(s,x)$. The time taken in this case is dominated by searching u and v in $\mathcal{R}_2(xy)$, that is $\tilde{O}(1)$.

Thus, the total query time of our algorithm is $\tilde{O}(1)$, and we can return the minimum of replacement paths found in Step 2 and 3 as our final answer. The reader can check that the space taken by our algorithm for a vertex t is $\tilde{O}(\sqrt{n\sigma})$. Thus the total space taken by our algorithm is $\tilde{O}(\sigma^{1/2}n^{3/2})$. Thus we have proved the main result, that is Theorem 1 of our paper.

References

[1] Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. Dynamic DFS in Undirected Graphs: breaking the O(m) barrier. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 730–739, 2016.

[2] Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant subgraph for single source reachability: generic and optimal. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 509–518, 2016.

[3] Aaron Bernstein. A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 742–755. Society for Industrial and Applied Mathematics, 2010.

[4] Aaron Bernstein and David Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 101–110. ACM, 2009.

[5] Davide Bilò, Keerti Choudhary, Luciano Gualà, Stefano Leucci, Merav Parter, and Guido Proietti. Efficient oracles and routing schemes for replacement paths. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 13:1–13:15, 2018.

[6] Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Compact and fast sensitivity oracles for single-source distances. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 13:1–13:14, 2016.

[7] Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 18:1–18:14, 2016.

[8] Greg Bodwin, Fabrizio Grandoni, Merav Parter, and Virginia Vassilevska Williams. Preserving distances in very faulty graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 73:1–73:14, 2017.

[9] Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.

[10] Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 506–515, 2009.

[11] Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 748–757. IEEE, 2012.

[12] Manoj Gupta and Shahbaz Khan. Multiple source dual fault tolerant BFS trees. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 127:1–127:15, 2017.

[13] John Hershberger and Subhash Suri. Vickrey prices and shortest paths: What is an edge worth? In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 252–259, 2001.

[14] Neelesh Khanna and Surender Baswana. Approximate shortest paths avoiding a failed vertex: Optimal size data structures for unweighted graphs. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, pages 513–524, 2010.

[15] Kavindra Malik, Ashok K Mittal, and Santosh K Gupta. The k most vital arcs in the shortest path problem. *Operations Research Letters*, 8(4):223–227, 1989.

[16] Merav Parter. Dual failure resilient BFS structure. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 481–490, 2015.

[17] Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 779–790, 2013.

[18] Liam Roditty. On the k-simple shortest paths problem in weighted directed graphs. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 920–928. Society for Industrial and Applied Mathematics, 2007.

[19] Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Transactions on Algorithms (TALG)*, 8(4):33, 2012.

[20] Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

[21] Oren Weimann and Raphael Yuster. Replacement paths via fast matrix multiplication. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 655–662. IEEE, 2010.

[22] Virginia Vassilevska Williams. Faster replacement paths. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1337–1346. SIAM, 2011.