Distance Sensitivity Oracles with Subcubic Preprocessing Time and Fast Query Time*

Shiri Chechik
shiri.chechik@gmail.com
Tel Aviv University
Israel

Sarel Cohen
sarelcoh@post.tau.ac.il
Tel Aviv University
Israel

ABSTRACT

We present the first distance sensitivity oracle (DSO) with subcubic preprocessing time and poly-logarithmic query time for directed graphs with integer weights in the range $[-M, M]$.

Weimann and Yuster [FOCS 10] presented a distance sensitivity oracle for a single vertex/edge failure with subcubic preprocessing time of $O(Mn^{\omega+1-\alpha})$ and subquadratic query time of $\widetilde{O}(n^{1+\alpha})$, where α is any parameter in $[0, 1]$, n is the number of vertices, m is the number of edges, the $\widetilde{O}(\cdot)$ notation hides poly-logarithmic factors in n and $\omega < 2.373$ is the matrix multiplication exponent.

Later, Grandoni and Vassilevska Williams [FOCS 12] substantially improved the query time to sublinear in n. In particular, they presented a distance sensitivity oracle for a single vertex/edge failure with $\widetilde{O}(Mn^{\omega+1/2} + Mn^{\omega+\alpha(4-\omega)})$ preprocessing time and $\widetilde{O}(n^{1-\alpha})$ query time.

Despite the substantial improvement in the query time, it still remains polynomial in the size of the graph, which may be undesirable in many settings where the graph is of large scale. A natural question is whether one can hope for a distance sensitivity oracle with subcubic preprocessing time and very fast query time (of poly-logarithmic in n).

In this paper we answer this question affirmatively by presenting a distance sensitive oracle supporting a single vertex/edge failure in subcubic $\widetilde{O}(Mn^{2.873})$ preprocessing time for $\omega = 2.373$, $\widetilde{O}(n^{2.5})$ space and near optimal query time of $\widetilde{O}(1)$.

For comparison, with the same $\widetilde{O}(Mn^{2.873})$ preprocessing time the DSO of Grandoni and Vassilevska Williams has $\widetilde{O}(n^{0.693})$ query time. In fact, the best query time their algorithm can obtain is $\widetilde{O}(Mn^{0.385})$ (with $\widetilde{O}(Mn^3)$ preprocessing time).

CCS CONCEPTS

• **Theory of computation → Shortest paths**; *Dynamic graph algorithms*; *Data structures design and analysis.*

*This publication is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803118 UncertainENV)

KEYWORDS

Distance Sensitivity Oracles, Replacement Paths, Fault-Tolerant, Shortest Paths

ACM Reference Format:
Shiri Chechik and Sarel Cohen. 2020. Distance Sensitivity Oracles with Subcubic Preprocessing Time and Fast Query Time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC '20), June 22–26, 2020, Chicago, IL, USA.* ACM, New York, NY, USA, 14 pages.
https://doi.org/10.1145/3357713.3384253

1 INTRODUCTION

Resilience to failures is an indispensable issue in modern networks. A failure event involves with some of the network's vertices or edges to be temporarily unavailable. This may cause some pre-computed structural information, such as distances, connectivity, flow, and so on, to be no longer valid.

In this paper we concentrate on maintaining distances in the presence of one edge (or vertex) failure in directed weighted graphs.

Our goal is to preprocess a graph $G = (V, E)$ in relatively short time and produce a data structure that can later answer subsequent queries of the form (s, t, e), where $s, t \in V$ and $e \in E$. The answer to such a query is the distance from s to t in the graph $G \setminus \{e\}$ (the graph obtained from G by discarding the edge e). We call such a data structure a distance sensitivity oracle, or a DSO.

Existing Oracles. For the case of a single edge (or vertex) failure, Demetrescu *et al.* [13] showed that it is possible to preprocess a directed weighted graph in $\widetilde{O}(mn^2)$ time to compute a data structure of size $O(n^2 \log n)$ capable of answering distance queries in $O(1)$ time. In two consecutive papers, Karger and Bernstein improved the preprocessing time of [13], first to $O(n^2\sqrt{m})$ [7] and later to $\widetilde{O}(mn)$ [1]. The size and the query time remain unchanged.

Duan and Pettie [14] considered the more involved case of two failures and presented an oracle with $O(n^2 \log^3 n)$ size, $O(\log n)$ query time and polynomial construction time.

The distance sensitivity oracle problem was also considered in the approximate regime (see e.g. [9, 10, 17]) and for special families of graphs (see e.g. [2, 3, 5, 8, 11, 12]).

Although this problem has been extensively studied and some of the bounds seem to be close to optimal by now, all the above mentioned algorithms for general graphs require at least $\Omega(n^3)$ preprocessing time for dense graphs. This may make these algorithms inadequate for many settings with large scale networks.

In the static regime, one can beat the $\Omega(n^3)$ bound in the case of integer weights of small absolute value (see [19, 24]). The question of the existence of distance sensitivity oracles with subcubic preprocessing time and subquadratic query time was asked by Weimann and Yuster [21]. They presented, for any parameter $\alpha \in [0,1]$, a distance sensitivity oracle with $O(Mn^{\omega+1-\alpha})$ preprocessing time (here $\omega < 2.373$ is the matrix multiplication exponent [18, 23]) and $\widetilde{O}(n^{1+\alpha})$ query time. More precisely, they presented an algorithm that can handle up to $O(\log n/\log\log n)$ edges or vertices failures but with query time that deteriorates as the number f of failures gets larger (that is, $\widetilde{O}(n^{2-(1-\alpha)/f})$ query time and $O(Mn^{\omega+1-\alpha})$ preprocessing time).

Recently, van den Brand and Saranurak [20] presented the first distance sensitive oracle that can handle $f \geq \log n$ updates (where an update is an edge insertion or deletion), with $\widetilde{O}(Mn^{\omega+(3-\omega)\mu})$ preprocessing time, $\widetilde{O}(Mn^{2-\mu}f^2+Mnf^{\omega})$ update time, and $\widetilde{O}(Mn^{2-\mu}f+Mnf^2)$ query time, where the parameter $\mu \in [0,1]$ can be chosen. When $M = \widetilde{O}(1)$, their DSO simultaneously improves the preprocessing time, update time and query time of Weimann and Yuster [21]. Note that their query time is at least linear.

In [15], Grandoni and Vassilevska Williams substantially improved the query time to sublinear in n for the case of a single edge failure. In particular, they presented a distance sensitivity oracle with $\widetilde{O}(Mn^{\omega+1/2} + Mn^{\omega+\alpha(4-\omega)})$ preprocessing time and $\widetilde{O}(n^{1-\alpha})$ query time. This is a huge improvement in the query time.

However, the query time still remains polynomial in the size of the graph, which may be undesirable in many settings where the graph is of large scale. This is especially important as often when designing distance oracles, the requirement is to have a very fast query time. A natural question is whether one can hope for a distance sensitivity oracle with subcubic preprocessing time and very fast query time (of poly-logarithmic in n). We answer this question affirmatively.

Our results. In this paper we present the first distance sensitivity oracle with subcubic preprocessing time and very fast query time for directed graphs with integer weights of absolute value bounded by M. Our result is summarized in the following theorem.

Theorem 1. *Given a weighted directed graph G with integer weights in the range $[-M,M]$, for $\omega \in [2.35, 2.373]$* [1] *one can construct w.h.p. (with probability of at least $1-1/n^Q$ for any constant $Q > 0$) a DSO supporting one edge failure in subcubic $\widetilde{O}(Mn^{\omega+1/2})$ time. The size of the data-structure is $\widetilde{O}(n^{2.5})$. Given a query (s,t,e), the DSO returns w.h.p. the distance $d_G(s,t,e)$ in $\widetilde{O}(1)$ time, and the replacement path $P(s,t,e)$ is returned in time proportional to the number of its edges $\widetilde{O}(|P(s,t,e)|)$.*

Our DSO has $\widetilde{O}(Mn^{2.873})$ preprocessing time for $\omega = 2.373$ and $\widetilde{O}(1)$ query time. For comparison, with the same $\widetilde{O}(Mn^{2.873})$ preprocessing time the DSO of Grandoni and Vassilevska Williams has $\widetilde{O}(n^{0.693})$ query time. In fact, the best query time their algorithm can obtain is $\widetilde{O}(Mn^{0.385})$ (with $\widetilde{O}(Mn^3)$ preprocessing time).

Notice that our preprocessing algorithm is subcubic for small integer weights in the range $[-M,M]$. More precisely, for $\omega = 2.373$ the preprocessing algorithm is subcubic when $M = o(n^{2.5-\omega}) = o(n^{0.127})$.

[1]We note that our running time is $\widetilde{O}(Mn^{\omega+1/2})$ for any $\omega \in [2.35, 2.373]$. Our running time consists of several factors, some of which are dominated by $\widetilde{O}(Mn^{\omega+1/2})$ for any $\omega \in [2.35, 2.373]$. For smaller ω our running time becomes more complicated.

In this paper for simplicity, we consider only edge failures, but we note that in the directed case, edge failures subsume vertex failures. This is a consequence of the following well known reduction. Replace every vertex v with two vertices v_{in} and v_{out}, and connect them by a direct edge (v_{in}, v_{out}). In addition, move all incoming edges to v to v_{in} and all outgoing edges from v to v_{out}. Now the failure of the edge (v_{in}, v_{out}) has the same effect as the failure of the vertex v.

In our construction we essentially show a subcubic reduction from DSO to the problem of computing a single distance for every pair of vertices s and t, that is, the distance from s and t avoiding $P^R_G(s,t)$ (where $P^R_G(s,t)$ is the subpath of the shortest path from s to t not including the first and last R edges) for some parameter R. Computing these n^2 distances is the most technical and challenging part of our algorithm.

Loosely speaking, in order to compute these n^2 distances we sample a small set B of vertices that w.h.p. hits the detour part of every replacement path $P(s,t,P^R_G(s,t))$ (where $P(x,y,S)$ for a pair of vertices x and y and a subset S of vertices is the shortest path between s and t in the graph $G \setminus S$). Then, for every vertex $v \in B$ and a destination vertex t (independent to s) we compute some replacements paths such that given s and t one can quickly pick one of these replacement paths in order to compute $P(v,t,P^R_G(s,t))$. Similarly, the algorithm computes $P(s,v,P^R_G(s,t))$. The algorithm then uses all these paths to deduce $P(s,t,P^R_G(s,t))$.

Unique shortest paths is a desired property in many algorithms and applications. This is also the case in our algorithm. However, it is not known how to achieve unique shortest paths in subcubic time. We present a subcubic algorithm that computes unique shortest paths whenever the paths contains more than R edges (and handles separately the case where the shortest path contains at most R edges).

Our generalization to negative weights is substantially more complicated as outlined in this paper, and described in detail in the full version.

2 PRELIMINARIES

Let $H = (V,E)$ be a weighted graph with integer edge weights in the range $[-M,M]$. Let P be a path in H. We denote by $w(P)$ the length of the path P which is defined as the sum of weights of the edges along P, and by $|P|$ the number of edges of P.

Let $u,v \in V$ be two vertices. We denote by $P_H(u,v)$ a shortest path from u to v in H, and by $d_H(u,v)$ the distance from u to v in the graph H (*i.e.*, $d_H(u,v) = w(P_H(u,v))$). When H is clear from the context, we sometimes abbreviate $P_H(u,v) = P(u,v)$ and $d_H(u,v) = d(u,v)$. Let $e = (x,y) \in E$, we define $d_H(s,e) = \min\{d_H(s,x), d_H(s,y)\}$.

Let $F \subset E \cup V$ be a set of edges and/or vertices, we denote by $H \setminus F$ the graph obtained by removing the set F of edges and vertices (along with their incident edges) from H.

Let $s,t \in V$ be two vertices and $e \in P_H(s,t)$ be an edge on the shortest path from s to t in the graph H. The replacement path associated with the triple (s,t,e), denoted by $P_H(s,t,e)$, is a shortest path from the *source* vertex s to the *target* vertex t avoiding the edge e in the graph H. We denote by $d_H(s,t,e) = w(P_H(s,t,e))$ the distance from s to t in the graph $H \setminus \{e\}$. Similarly, let $F \subset E \cup V$ be a subset of edges and/or vertices, we denote by $P_H(s,t,F)$ a shortest path from s to t in the graph $H\backslash F$ and by $d_H(s,t,F) = w(P_H(s,t,F))$ the distance from s to t in the graph $H \setminus F$.

For a graph G' we denote by $V(G')$ the set of its vertices, and by $E(G')$ the set of its edges.

Given a path P that contains the vertices $u, v \in V$ such that u appears before v along P, we denote by $P[u,v]$ the subpath of P from u to v. Let $e \in E, v \in V$ such that e appears before v along P, we denote by $P[e,v]$ the subpath of P from e to v (including the edge e).

We now define the path concatenation operator $\circ$. Let $P_1 = (x_1, x_2, \ldots, x_r)$ and $P_2 = (y_1, y_2, \ldots, y_t)$ be two paths. $P = P_1 \circ P_2$ is defined as the path $P = (x_1, x_2, \ldots, x_r, y_1, y_2, \ldots, y_t)$, and it is well defined if either $x_r = y_1$ or $(x_r, y_1) \in E$.

A distance sensitivity oracle (abbreviated DSO) is a space efficient data-structure, which preprocesses a graph H, such that given a query (s,t,e), computes efficiently the distance $d_H(s,t,e)$ from s to t in the graph $H \setminus \{e\}$ and a replacement path $P_H(s,t,e)$.

The Single-Source Replacement Paths (SSRP) problem is defined as follows. Given a fixed source vertex s in the graph H the $\mathrm{SSRP}_H(s)$ problem is to compute the distances $d_H(s,t,e)$ and replacement paths $P_H(s,t,e)$ for every vertex $t \in V$ and for every edge $e \in P_H(s,t)$.

Given $G = (V,E)$, we denote by $G^T = (V, E^T)$ the graph G with reversed edge directions, *i.e.*, $E^T = \{(v,u) \mid (u,v) \in E\}$. Given an edge $e = (u,v)$ we denote by $e^T = (v,u)$.

It is well known that given (s,t,e) there exists a replacement path $P(s,t,e)$ that is composed of a common prefix $\mathrm{CommonPref}(P(s,t,e), P(s,t))$ with the shortest path $P(s,t)$, a detour $\mathrm{Detour}(P(s,t,e), P(s,t))$ which is disjoint from the shortest path $P(s,t)$, and finally a common suffix $\mathrm{CommonSuff}(P(s,t,e), P(s,t))$ which is common with the shortest path $P(s,t)$. Therefore, it holds that
$P(s,t,e) = \mathrm{CommonPref}(P(s,t,e), P(s,t)) \circ \mathrm{Detour}(P(s,t,e), P(s,t)) \circ \mathrm{CommonSuff}(P(s,t,e), P(s,t))$.

The following sampling Lemma is a folklore.

Lemma 2. *[Random Sampling, proof in the full version] Let G be a graph with n vertices of which R vertices are red and $n - R$ vertices are blue. By choosing a random set of $\widetilde{O}(n/R)$ vertices (more precisely, $nQ \ln n/R$ expected number of vertices), at least one of the chosen vertices is red with high probability (with probability at least $1 - 1/n^Q$ for any constant $Q > 0$).*

Given a rooted tree T containing n vertices, the Least Common Ancestor of a pair of vertices u and v ($\mathrm{LCA}(u,v)$) is the lowest (*i.e.* farthest from the root) vertex that has both u and v as descendants, where we define each vertex to be a descendant of itself (so if u is the direct parent of v then u is the lowest common ancestor). Bender and Farach-Colton [6] presented a simple LCA data-structure that preprocesses T in linear time and answers LCA queries in constant time.

Lemma 3. *[See [6]] Given a rooted tree T containing n vertices, one can construct an LCA data-structure in linear $O(n)$ time and answer LCA queries in constant time.*

We next list several previous algorithms and data-structures we use in our construction.

Lemma 4 (See [24]). *Given a directed graph with integer weights in the range $[-M,M]$, APSP can be computed in $\widetilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$ time.*

Definition 1. *[The distance $d^{\leq R}(s,t)$ and path $P^{\leq R}(s,t)$] Let $G = (V,E)$ be a weighted graph, let $s,t \in V$ and let $R > 0$ be an integer parameter. Define $d^{\leq R}(s,t)$ to be the length of the shortest s-to-t path on at most R edges, and $P^{\leq R}(s,t)$ is one such path. If there is no path from s to t containing at most R edges then set $d^{\leq R}(s,t) = \infty$.*

In the following lemma, we describe how to compute $APSP^{\leq R}(G)$, that computes for every pair of vertices $s,t \in V$ the length of the shortest s-to-t path on at most R edges. Note that our requirement from the algorithm $APSP^{\leq R}(G)$ is different than the algorithm in Corollary 1 in [15]. In [15] the algorithm computes for every $s,t \in V$ an estimate of the distance $d(s,t)$ that is correct w.h.p. if there exists a shortest path from s to t on at most R edges. However, the path returned could potentially contain more than R edges. For our needs, it is important to actually find a path that contains at most R edges and that is shortest among all paths of at most R edges.

Lemma 5. *[See more details in the full version] There is an algorithm, hereafter referred to as $APSP^{\leq R}$, that computes distances and paths $\{d^{\leq R}(s,t), P^{\leq R}(s,t)\}_{s,t \in V}$ in $\widetilde{O}(R \cdot Mn^{\omega})$ time.*

The following lemmas were obtained by Grandoni and Vassilevska Williams in [15] and we use these lemmas extensively throughout the paper.

Lemma 6. *[See [15]] The SSRP algorithm computes, w.h.p., the distances $\{d(s,t,e)\}_{s,t \in V, e \in P(s,t)}$ in $\mathrm{SSRP}(M,n) = \widetilde{O}(Mn^{\omega})$ time for directed graphs with integer edges weights in $[1,M]$ and $\mathrm{SSRP}([-M,M],n) = \widetilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$ time for directed graphs with integer edges weights in $[-M,M]$.*

Lemma 7. *[See [15]] Given a directed graph H with integer edges weights in $[-M,M]$, let $1 \leq X \leq n$, a data-structure $\mathrm{DSO}_X(H)$ that given a query (s,t,e) computes w.h.p. $d(s,t,e)$ in $\widetilde{O}(n/X)$ time, can be constructed in $\widetilde{O}(Mn^{\omega} \cdot (X^{4-\omega} + \sqrt{n}))$ time.*

The following lemma was obtained by Weimann and Yuster in [21].

Lemma 8. *[See [21]] Let $1 \leq R \leq n$ be an integer and G be a directed graph with integer weights in the range $[-M,M]$. There exists a data-structure $\mathrm{ShortDSO}_R(G)$ supporting one edge/vertex failure, that given a query (s,t,e) the DSO returns a distance $\hat{d}(s,t,e)$ and a path $\hat{P}(s,t,e)$ such that it always holds that $\hat{d}(s,t,e) \geq d(s,t,e)$. Furthermore, if there exists a replacement path $P(s,t,e)$ from s to t in the graph $G \setminus \{e\}$ that contains at most $3R$ edges, then w.h.p. $\hat{d}(s,t,e) = d(s,t,e)$ and $\hat{P}(s,t,e)$ is a replacement path for (s,t,e). The preprocessing time is $\widetilde{O}(RM^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$ and query time is $\widetilde{O}(1)$.*

Note that for every pair of vertices $s,t \in V$, there may be many different shortest paths from s to t in the graph G. We define the graph $\tilde{G}$ as follows, such that shortest paths are unique in $\tilde{G}$.

Definition 2. *[The graph $\tilde{G}$ and the unique shortest paths $P_{\tilde{G}}(s,t)$] Let $\tilde{G}$ be the graph obtained from G by adding small perturbations to the weights of the edges (e.g., by adding to the weight of every edge a random real number in the range $[0, 1/n)$), such that w.h.p. for every* *pair of vertices $s, t \in V$, the shortest path $P_{\tilde{G}}(s,t)$ in $\tilde{G}$ is unique and $\lfloor d_{\tilde{G}}(s,t) \rfloor = d(s,t)$.*

3 OVERVIEW

In this section, we describe an overview of our DSO with subcubic $\widetilde{O}(M^{\frac{5-\omega}{9-2\omega}} n^{\frac{16+\omega-\omega^2}{9-2\omega}}) = \widetilde{O}(M^{0.62} n^{2.9953})$ preprocessing time (for $\omega = 2.373$) and $\widetilde{O}(1)$ query time for weighted directed graphs with positive integer weights in the range $[1, M]$. In Section 4, we generalize our DSO to handle negative weights as well and in Section 5 we improve the preprocessing time to $\widetilde{O}(Mn^{\omega+1/2}) = \widetilde{O}(Mn^{2.873})$ (for $\omega \in [2.35, 2.373]$).

Following, we describe the different cases we consider for the query (s, t, e) and then present a data structure for each case that quickly returns an estimation for $d(s, t, e)$ such that the estimation is always at least $d(s, t, e)$ and if we are in the right case then w.h.p the estimation is $d(s, t, e)$. Finally, the algorithm answers the query by returning the minimum estimated distance from all of these cases. Cases 0-4 are relatively easy to handle given the prior work of Grandoni and Vassilevska Williams [15] and Weimann and Yuster [21], where case 5 is the most technical and difficult part of our algorithm.

Let $1 \le R \le n$ be a parameter that we will set later on.

Case 0: There exists at least one shortest path from s to t in G that contains at most R edges. In other words, this case happens iff $d^{\le R}(s,t) = d(s,t)$ (see Definition 1 of $d^{\le R}(s,t)$).

Case 1: $e \notin P_{\tilde{G}}(s,t)$. (see Definition 2 of $P_{\tilde{G}}(s,t)$).

Case 2: e is among the first or last R edges of the unique shortest path $P_{\tilde{G}}(s,t)$.

Case 3: There exists a replacement path from s to t in $G \setminus \{e\}$ that contains at most $3R$ edges.

Case 4: There exists at least one replacement path $P(s,t,e)$ for the triple (s,t,e) such that its common prefix CommonPref$(P(s,t,e), P_{\tilde{G}}(s,t))$ with the unique shortest path $P_{\tilde{G}}(s,t)$ contains at least R edges, or its common suffix CommonSuff$(P(s,t,e), P_{\tilde{G}}(s,t))$ with the unique shortest path $P_{\tilde{G}}(s,t)$ contains at least R edges.

Case 5: The complement of the previous cases. The triple (s,t,e) belongs to Case 5 if it does not belong to any of the cases 0-4. We refer to Case 5 as the R-critical case, and a triple (s,t,e) that belongs to Case 5 is called an R-critical query.

Handling R-critical queries is the most difficult part in our DSO, and it is our main technical contribution. Let us precisely define when a query/triple (s,t,e) is called an R-critical query.

Definition 3. *Given a path $P = \langle v_0, \ldots, v_k \rangle$ and a positive integer $R > 0$, we denote by $P^R := \langle v_R, \ldots, v_{k-R} \rangle$ the subpath of P obtained by removing the first and last R edges of P. If $2R \ge |P|$ then P^R is the empty path.*

Definition 4. *A query/triple (s,t,e) with $s,t \in V, e \in E$ is called R-critical if the following conditions hold. Not case 0: every shortest path from s to t in G contains more than R edges. Not case 1: $e \in P_{\tilde{G}}(s,t)$. Not case 2: $e \in P^R_{\tilde{G}}(s,t)$, i.e., e is not one of the first or last R edges of $P_{\tilde{G}}(s,t)$. This also implies that $P_{\tilde{G}}(s,t)$ contains at least $2R+1$ edges. Not case 3: every replacement path $P(s,t,e)$ for the triple (s,t,e) contains more than $3R$ edges. Not case 4: every replacement path $P(s,t,e)$ for the triple (s,t,e) satisfies that the prefix* CommonPref$(P(s,t,e), P_{\tilde{G}}(s,t))$ *contains less than R edges, and the suffix* CommonSuff$(P(s,t,e), P_{\tilde{G}}(s,t))$ *contains less than R edges.*

3.1 Cases 0-4

In this section, we describe how the algorithm handles Cases 0-4.

Using the uniqueness of the shortest paths as a new tool. Having unique shortest paths in the input graph is an essential property for our algorithm. However, the given graph G might not satisfy this requirement. A common trick to achieve unique shortest paths is obtaining a graph $\tilde{G}$ by adding small random perturbations to the edges of G and then break ties according to $\tilde{G}$. This approach is problematic in our case as edges in $\tilde{G}$ are not anymore integers so computing APSP on such a graph might be too expensive and impossible to do in truly subcubic time (even if somehow we scale the edge weights in $\tilde{G}$ to be integers then the maximum edge weight would be too large and computing APSP would be too expensive again). To overcome this issue, we develop an algorithm that computes in subcubic time shortest paths in $\tilde{G}$ for every pair of vertices that their shortest path contains at least R edges for some parameter R. We use the APSP$^{\le R}$ algorithm described in Lemma 5 that computes distances and shortest paths on at most R edges $\{d^{\le R}(s,t), P^{\le R}(s,t)\}_{s,t \in V}$ in $\widetilde{O}(R \cdot Mn^{\omega})$ time. Below we describe how to handle the case where the shortest path from s to t is on at most R edges (Case 0) and for the rest of the cases, when $d^{\le R}(s,t) > d(s,t)$, we assume (loosely speaking) we have unique shortest paths as described hereinafter. In other words, for every $s,t \in V$ such that $d^{\le R}(s,t) > d(s,t)$ the algorithm finds w.h.p. the unique shortest path $P_{\tilde{G}}(s,t)$ as follows.

The partial shortest paths trees $\{\tilde{T}_s\}_{s \in V}$. The algorithm constructs partial shortest paths trees $\{\tilde{T}_s\}_{s \in V}$ such that w.h.p. $P_{\tilde{G}}(s,t)$ is the path in $\tilde{T}_s$ from s to t when $d^{\le R}(s,t) > d(s,t)$. Loosely speaking, the set $\{\tilde{T}_s\}_{s \in V}$ is a set of shortest paths trees in $\tilde{G}$ for shortest paths that contain more than R edges. We briefly describe how to construct the partial shortest paths trees $\{\tilde{T}_s\}_{s \in V}$.

Let $B \subseteq V$ be a subset of vertices obtained by choosing every vertex independently uniformly at random with probability $\frac{Q \ln n}{R}$ for large enough constant $Q > 0$. From every vertex $v \in B$ the algorithm runs Dijkstra in the graph $\tilde{G}$ and obtains the unique shortest paths trees $\{\tilde{T}_v\}_{v \in B}$ (also run Dijkstra from v in the graph $\tilde{G}^T$ with reversed edge directions).

Given the complete shortest paths trees $\{\tilde{T}_v\}_{v \in B}$, the algorithm constructs the partial shortest paths trees $\{\tilde{T}_v\}_{v \in V}$ as follows. For every $s \in V \setminus B$ the algorithm initializes $\tilde{T}_s$ as a tree containing one vertex which is the root s. Next, for every vertex $t \in V$ such that $d^{\le R}(s,t) > d(s,t)$ the algorithm scans all the vertices $v \in B$ and finds a vertex $v = \arg\min_{v \in B}\{d_{\tilde{G}}(s,v) + d_{\tilde{G}}(v,t)\}$. The path $P_{\tilde{G}}(s,t)$ is w.h.p. the path from s to t in $\tilde{T}_v$. Then, the algorithm scans this path from t towards s in $\tilde{T}_v$. As the algorithm scans this t-to-s path, the algorithm adds the subpath that it scans to the tree $\tilde{T}_s$, until it reaches a vertex that was already previously added to $\tilde{T}_s$ and then it stops the scan of the path from t towards s. It is not difficult to prove that the algorithm takes $\widetilde{O}(n^3/R)$ time with high probability.

Handling Case 0. In this case there exists a shortest path from s to t in G that contains at most R edges. During preprocessing, the algorithm constructs the $\mathrm{DSO}_X(G)$ data-structure of Grandoni and Vassilevska Williams [15] according to Lemma 7 and then uses it to compute the distance $d(s,t,e)$ for every $s,t \in V$ such that $d^{\leq R}(s,t) = d(s,t)$ and for every edge $e \in P^{\leq R}(s,t)$ (there are at most R edges in $P^{\leq R}(s,t)$). The algorithm queries the oracle $\mathrm{DSO}_X(G)$ with (s,t,e) according to Lemma 7 and stores the answer in a hash table h_0. Given a query (s,t,e) the algorithm checks whether or not (s,t,e) is a key of the hash table h_0. If so, the algorithm returns $h_0[s,t,e]$ as the answer to the query (s,t,e). According to Lemma 7 it holds that $h_0[s,t,e] \geq d(s,t,e)$ and w.h.p. $h_0[s,t,e] = d(s,t,e)$. Next, we analyse the construction time of the hash table h_0. As there are $O(n^2 R)$ triples $(s,t,e) \in V \times V \times E$ such that $e \in P^{\leq R}(s,t)$, and computing the distance $d(s,t,e)$ using $\mathrm{DSO}_X(G)$ takes $\widetilde{O}(n/X)$ time w.h.p. according to Lemma 7, then constructing the hash table h_0 takes $T_{\text{construct}}(\mathrm{DSO}_X(G)) + \widetilde{O}(n^3 R/X)$ time. Note that according to Lemma 7, $T_{\text{construct}}(\mathrm{DSO}_X(G)) = \widetilde{O}(Mn^{\omega} \cdot (X^{4-\omega} + \sqrt{n}))$.

Handling Case 1. It is relatively easy to check whether or not $e \in P_{\tilde{G}}(s,t)$ using an LCA data-structure as in Lemma 3 by, loosely speaking, checking whether or not e is an ancestor of t in $\tilde{T}_s$ (see the full-version for more details). If $e \notin P_{\tilde{G}}(s,t)$ the algorithm returns $d(s,t)$ as the distance from s to t in $G \setminus \{e\}$. For the rest of this section assume $e \in P_{\tilde{G}}(s,t)$. The preprocessing time of this step is $O(n^2)$, as it takes $O(n)$ time to construct an LCA data-structure of a single tree $\tilde{T}_s$ as in Lemma 3 and the algorithm constructs LCA data-structures for all the trees $\{\tilde{T}_s\}_{s \in V}$.

Handling Case 2. During preprocessing, the algorithm uses the $\mathrm{DSO}_X(G)$ data-structure according to Lemma 7 to compute the distance $d(s,t,e)$ for every $s,t \in V, e \in E$ such that e is among the first or last R edges of $P_{\tilde{G}}(s,t)$, and stores all the computed distances in a hash table h_2. Since the $\mathrm{DSO}_X(G)$ data-structure was already computed during the construction of h_0 above, it follows that the time for constructing h_2 is $\widetilde{O}(n^3 R/X)$, as there are $O(n^2 R)$ triples (s,t,e) where $s,t \in V$ and e is among the first or last R edges of $P_{\tilde{G}}(s,t)$, and according to Lemma 7 computing $d(s,t,e)$ for each such triple takes w.h.p. $\widetilde{O}(n/X)$ time using $\mathrm{DSO}_X(G)$. Given a query (s,t,e) the algorithm checks whether or not (s,t,e) is a key of the hash table h_2. If so, the algorithm returns $h_2[s,t,e]$ as the answer to the query (s,t,e). According to Lemma 7 it holds that $h_2[s,t,e] \geq d(s,t,e)$ and w.h.p. $h_2[s,t,e] = d(s,t,e)$.

Handling Case 3. To handle the case that there exists a replacement path from s to t in $G \setminus \{e\}$ that contains at most $3R$ edges, the algorithm constructs the oracle $\mathrm{ShortDSO}_{3R}(G)$ of Weimann and Yuster [21] as in Lemma 8. Given a query (s,t,e) the algorithm sets $d_3(s,t,e)$ as the answer of $\mathrm{ShortDSO}_{3R}(G)$ to the query (s,t,e). By Lemma 8, $d_3(s,t,e) \geq d(s,t,e)$ and if there exists a replacement path $P(s,t,e)$ from s to t in the graph $G \setminus \{e\}$ that contains at most $3R$ edges, then w.h.p. $d_3(s,t,e) = d(s,t,e)$. The preprocessing procedure of this step takes $\widetilde{O}(T_{\text{construct}}(\mathrm{ShortDSO}_{3R}(G)) = \widetilde{O}(RM^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$ time, where the last equality holds from Lemma 8.

Handling Case 4. During preprocessing, for every vertex $v \in B$ (the set B as described above) compute $\mathrm{SSRP}_G(v)$ according to Lemma 6 in the graph G and $\mathrm{SSRP}_{G^T}(v)$ in the graph G^T with reverse edge directions. For every $t \in V, e \in E$ denote by $\mathrm{SSRP}_G(v,t,e)$ the estimated distance from v to t in $G \setminus \{e\}$ as computed by $\mathrm{SSRP}_G(v)$. Next, for every $s,t \in V$ the algorithm scans the path $P_{\tilde{G}}(s,t)$ from s towards t (and from t towards s) until it finds the first vertex v_s (and the last vertex v_t) of $P_{\tilde{G}}(s,t)$ that belongs to B (if any such vertex exists). During the query, the algorithm computes $d_4(s,t,e) = \min\{d(s,v_s) + \mathrm{SSRP}_G(v_s,t,e), \mathrm{SSRP}_{G^T}(v_t,s,e^T) + d(v_t,t)\}$. If there exists at least one replacement path $P(s,t,e)$ for the triple (s,t,e) such that its common prefix $\mathrm{CommonPref}(P(s,t,e), P_{\tilde{G}}(s,t))$ with the unique shortest path $P_{\tilde{G}}(s,t)$ contains at least R edges then w.h.p. v_s is on the prefix part of $P(s,t,e)$ and then it holds w.h.p. that $d_4(s,t,e) = d(s,v_s) + d(v_s,t,e) = d(s,t,e)$. Similarly, if there exists at least one replacement path $P(s,t,e)$ for the triple (s,t,e) such that its common suffix $\mathrm{CommonSuff}(P(s,t,e), P_{\tilde{G}}(s,t))$ with the unique shortest path $P_{\tilde{G}}(s,t)$ contains at least R edges then w.h.p. v_t is on the suffix part of $P(s,t,e)$ and then it holds w.h.p. that $d_4(s,t,e) = d(s,v_t,e) + d(v_t,t) = d(s,t,e)$. It follows that if (s,t,e) belongs to Case 4 (and not to Cases 0 or 2) then w.h.p. $d_4(s,t,e) = d(s,t,e)$. The preprocessing procedure for handling Case 4 takes $\widetilde{O}(\frac{n}{R} \cdot \mathrm{SSRP}(M,n) + n^2 R) = \widetilde{O}(\frac{n}{R} \cdot Mn^{\omega} + n^2 R)$ time, where the last equality holds from Lemma 6.

3.2 Handling Case 5 - an Overview

Roughly speaking Case 5 is when the edge $e \in P^R_{\tilde{G}}(s,t)$, every replacement path is relatively long (at least $3R$ edges) and for every replacement path the prefix and suffix parts are relatively short (at most R edges). We say that (s,t,e) is R-critical if it belongs to case 5.

In order to handle all R-critical queries, we first observe an important property of all R-critical queries (s,t,e), that is, all R-critical queries for fixed $s,t \in V$ have the same answer $d_G(s,t,P^R_{\tilde{G}}(s,t))$ (see Figure 1). This follows pretty easily from the fact that for all of these queries (s,t,e), we have that $e \in P^R_{\tilde{G}}(s,t)$ and every replacement of (s,t,e) has short prefix and suffix and therefore avoids $P^R_{\tilde{G}}(s,t)$ (thus in particular bypass all edges e' such that (s,t,e') is also R-critical).

$$\boldsymbol{d(s,t,e_1) = d(s,t,e_2) = \ldots = d(s,t,e_\ell) = d(s,t,P^R_{\tilde{G}}(s,t))}$$

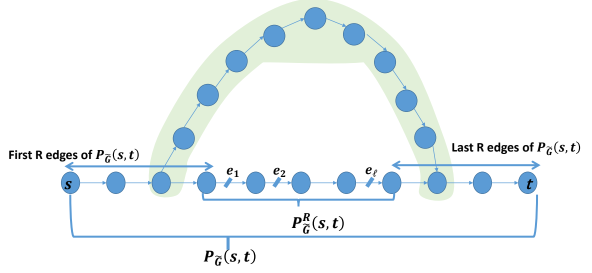
First R edges of $P_{\tilde{G}}(s,t)$
Last R edges of $P_{\tilde{G}}(s,t)$
s
e_1
e_2
e_ℓ
t
$P^R_{\tilde{G}}(s,t)$
$P_{\tilde{G}}(s,t)$

Figure 1: All R-critical queries $(s,t,e_1), (s,t,e_2), \ldots, (s,t,e_\ell)$ can be answered with the same distance $d(s,t,e_1) = d(s,t,e_2) = \ldots = d(s,t,e_\ell) = d(s,t,P^R_{\tilde{G}}(s,t))$. We denote by $P^R_{\tilde{G}}(s,t)$ the subpath of $P_{\tilde{G}}(s,t)$ obtained by discarding the first and last R edges of it.

Lemma 9. *[See Figure 1] Let $s, t \in V$, $e \in P_{\tilde{G}}(s,t)$ such that (s,t,e) is R-critical. Then $d_G(s,t,e) = d_G(s,t,P^R_{\tilde{G}}(s,t))$.*

Proof. We first prove that $d_G(s,t,e) \le d_G(s,t,P^R_{\tilde{G}}(s,t))$. Since (s,t,e) is R-critical, then by Definition 4 it holds that $e \in P^R_{\tilde{G}}(s,t)$, and thus $d_G(s,t,e) = d_{G\setminus\{e\}}(s,t) \le d_{G\setminus P^R_{\tilde{G}}(s,t)}(s,t) = d_G(s,t,P^R_{\tilde{G}}(s,t))$ as $G \setminus P^R_{\tilde{G}}(s,t) \subseteq G \setminus \{e\}$ and the distance from s to t may only increase when we remove more edges from $G \setminus \{e\}$.

For the other direction we prove $d_G(s,t,e) \ge d_G(s,t,P^R_{\tilde{G}}(s,t))$. Let $P_G(s,t,e)$ be a replacement path for (s,t,e). Since (s,t,e) is R-critical, then according to Definition 4 it holds that the prefix $\text{CommonPref}(P_G(s,t,e), P_{\tilde{G}}(s,t))$ contains less than R edges, and the suffix $\text{CommonSuff}(P_G(s,t,e), P_{\tilde{G}}(s,t))$ contains less than R edges. Thus, the replacement path $P_G(s,t,e)$ is disjoint from $P^R_{\tilde{G}}(s,t)$, and $P_G(s,t,e)$ is a path in the graph $G \setminus P^R_{\tilde{G}}(s,t)$ whose length is $d_G(s,t,e)$. Hence, $d_G(s,t,e) \ge d_G(s,t,P^R_{\tilde{G}}(s,t))$. □

This means that in order to handle Case 5, we only need to compute in the preprocessing stage the values $d_5(s,t) = d_G(s,t,P^R_{\tilde{G}}(s,t))$ for every $s,t \in V$. However, computing all these values in subcubic time turned out to be a non trivial task.

As described in the previous cases, the set B is a set of vertices of expected size $\widetilde{O}(n/R)$ and the algorithm computes the SSRP algorithm of Grandoni and Vassilevska Williams [15] from every vertex $v \in B$ in the graph G and in the graph G^T. Our goal is to find the distance $d_G(s,t,P^R_{\tilde{G}}(s,t))$ (and a shortest path $P(s,t,P^R_{\tilde{G}}(s,t))$). It is pretty easy to show that in R-critical queries the detour part contains more than R edges. To see this, if (s,t,e) is an R-critical query, then since Case 3 of Definition 4 does not hold then every replacement path for (s,t,e) contains more than $3R$ edges and since Case 4 of Definition 4 does not hold then both the prefix and suffix parts of every replacement path contain less than R edges and thus it follows that the detour part contains more than R edges. Hence, according to Lemma 2, w.h.p. the detour part contains a vertex v from B (see Figure 2). We would like to use the vertex v in order to compute $d_G(s,t,P^R_{\tilde{G}}(s,t))$. Unfortunately, we don't have the vertex v in advance and also we don't know in advance for which edges e, (s,t,e) is R-critical. If we would have magically given an edge e such that (s,t,e) is R-critical then by iterating all the vertices in B, we could have found such a vertex v that is on a detour part of some replacement path for (s,t,e). We could have then used e and v to compute $d_G(s,t,P^R_{\tilde{G}}(s,t))$. However, iterating over all edges $e \in P^R_{\tilde{G}}(s,t)$ and vertices $v \in B$ for every $s,t \in V$ is too expensive. Our next attempt is to estimate some distances between v and t only (i.e., independent of s) and similarly between v and s in the reverse graph G^T such that given s and t we can quickly (in poly-log time) find the distance $d_G(v,t,P^R_{\tilde{G}}(s,t))$ (and similarly $d_G(s,v,P^R_{\tilde{G}}(s,t))$). We will then estimate $d_G(s,t,P^R_{\tilde{G}}(s,t)) = \min_{v\in B}\{d_G(s,v,P^R_{\tilde{G}}(s,t)) + d_G(v,t,P^R_{\tilde{G}}(s,t))\}$. The tricky part is how to estimate the distance $d_G(v,t,P^R_{\tilde{G}}(s,t))$ as apriori we don't know s and need to be able to handle any $s \in V$.

Given an edge $e \in P_{\tilde{G}}(v,t) = \langle v = v_0, \dots, v_k = t\rangle$, we define the index $i_{\tilde{G}}(v,t,e)$ as the index of the first vertex of the detour part of some replacement path for (v,t,e). When v,t are known from the context, we abbreviate $i_e := i_{\tilde{G}}(v,t,e)$ and $d_e := d_G(v,t,e)$. In other words, v_{i_e} is the first vertex of the detour part of some replacement path for (v,t,e) and d_e is the length of such a replacement path. In Section 3.3 we prove the following lemma.

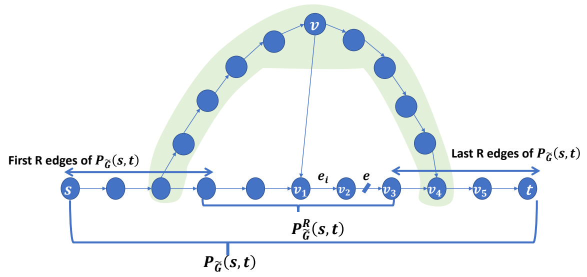

Figure 2: In this figure (s,t,e) is an R-critical query, $P(s,t,e)$ is an arbitrary replacement path for (s,t,e), the vertex v is on the detour part of $P(s,t,e)$, the path $P_{\tilde{G}}(v,t) = \langle v = v_0, v_1, \dots, v_5, v_6 = t\rangle$ is the unique shortest path from v to t in $\tilde{G}$, and $e_i = e_1 = (v_1,v_2)$ is the first common edge of $P_{\tilde{G}}(s,t)$ and $P_{\tilde{G}}(v,t)$ (which is not among the first R edges of $P_{\tilde{G}}(s,t)$).

Lemma 10. *[Proof in Section 3.3] One can compute a set of indices $\{i_{\tilde{G}}(v,t,e)\}_{v\in B, t\in V, e\in P_{\tilde{G}}(v,t)}$ in $\widetilde{O}(Mn^3/R)$ time.*

The algorithm constructs a table $\mathcal{T}_{v,t}(G)$ that consists of pairs (i_e, d_e) for some of the edges $e \in P_{\tilde{G}}(v, v_{k-R})$. This table will be used later on to compute $P(v,t,P^R_{\tilde{G}}(s,t))$ for a given vertex $s \in V$. The pair (i_e, d_e) in the table is as follows. $d_e = d_G(v,t,e)$ and the index i_e is the index of the first vertex in the detour part of some replacement path for (v,t,e). For every edge $e \in P_{\tilde{G}}(v, v_{k-R})$ such that all replacement paths $P(v,t,e)$ satisfy that the detour part enters $P_{\tilde{G}}(v,t)$ in the last R edges (we will later show how to check it), the algorithm adds the pair (i_e, d_e) to a table $\mathcal{T}_{v,t}(G)$. The algorithm then sorts the table according the first entry of the pair (i_e, d_e), i.e., according to i_e. We will later show the following. First, for every entry in the table (i_e, d_e) we have that $d_e = d(v,t,e) = d(v,t,P_{\tilde{G}}(v_{i_e}, v_{k-R}))$. Second, assume that there is a triple (s,t,e) such that (s,t,e) is R-critical and v is on the detour of some replacement path for (s,t,e) then (i_e, d_e) is an entry of the table such that $d_e = d(v,t,e)$ and i_e is the index of the first vertex of the detour part of some replacement path for (v,t,e). Third, the entries of the table are monotone (*i.e.*, if (i_e, d_e) and $(i_{e'}, d_{e'})$ are entries of the table such that $i_e \ge i_{e'}$ then $d_e \le d_{e'}$).

Next, we briefly explain how given a vertex s the algorithm uses the table to compute $P(v,t,P^R_{\tilde{G}}(s,t))$. Note that due to the uniqueness of shortest paths in $\tilde{G}$ we have that the paths $P_{\tilde{G}}(v,t)$ and $P_{\tilde{G}}(s,t)$ have some common suffix (and are disjoint up to this common suffix). Let $e_i = (v_i, v_{i+1}) \in P_{\tilde{G}}(v,t) = \langle v_0, \dots, v_k\rangle$ be the first common edge of $P_{\tilde{G}}(v,t)$ and $P_{\tilde{G}}(s,t)$ (see figure 2). Assume that e_i is not among the first or last R edges of $P_{\tilde{G}}(s,t)$ (we handle these cases in the full version, but for simplicity we omit it from this overview).

In order to estimate the distance $d_G(v,t,P^R_{\tilde{G}}(s,t))$ the algorithm does the following. The algorithm finds the maximum entry $(i_{e'}, d_{e'})$

in the table such that $i_{e'} \leq i$ (as the table is sorted then finding this pair can easily be done using a binary search) and returns $d'(v,t,P^R_{\tilde{G}}(s,t)) = d_{e'}$. If no such entry exists in the table the algorithm returns $d'(v,t,P^R_{\tilde{G}}(s,t)) = \infty$.

We want to show that the distance $d'(v,t,P^R_{\tilde{G}}(s,t))$ returned for the pair (s,t) is correct in the following sense. For every $e \in P^R_{\tilde{G}}(s,t)$ we have $d'(v,t,P^R_{\tilde{G}}(s,t)) \geq d(v,t,e)$. Second, if there exists an edge e such that (s,t,e) is R-critical and v is on the detour part of some replacement path for (s,t,e) then $d'(v,t,P^R_{\tilde{G}}(s,t)) = d(v,t,e)$.

To see the first requirement, if there is no entry $(i_{e'}, d_{e'})$ in the table $\mathcal{T}_{v,t}(G)$ such that $i_{e'} \leq i$ then the algorithm returns $d'(v,t,P^R_{\tilde{G}}(s,t)) = \infty$ and it trivially holds that $d'(v,t,P^R_{\tilde{G}}(s,t)) \geq d(v,t,e)$. Otherwise, the algorithm returns $d_{e'}$ such that $(i_{e'}, d_{e'})$ is an entry of the table and $i_{e'} < i$. Let $P(v,t,e')$ be the replacement path that the entry $(i_{e'}, d_{e'})$ represents. By the properties of the table we mentioned above we have that $P(v,t,e')$ avoids $P_{\tilde{G}}(v_{i_{e'}}, v_{k-R})$. If $e \in P_{\tilde{G}}(v,t)$ then we show that $P(v,t,e')$ avoids also e and therefore $d_{e'} \geq d(v,t,e)$. To see that, observe that $e \in P_{\tilde{G}}(v_i, v_{k-R})$ (as $e_i = (v_i, v_{i+1})$ is the first common edge of $P_{\tilde{G}}(v,t)$ and $P_{\tilde{G}}(s,t)$ and e is a common edge of $P^R_{\tilde{G}}(s,t)$ and $P_{\tilde{G}}(v,t)$). Furthermore as $i_{e'} < i$ then e appears along the path $P_{\tilde{G}}(v,t)$ between $v_{i_{e'}}$ and v_{k-R}, and since $P(v,t,e')$ avoids $P_{\tilde{G}}(v_{i_{e'}}, v_{k-R})$ then $P(v,t,e')$ avoids also e. Otherwise $e \notin P_{\tilde{G}}(v,t)$ and the distance the algorithm returns is a distance of some path from v to t and therefore at least $d(v,t)$. We get $d_{e'} \geq d(v,t) = d(v,t,e)$, as required.

We are left showing the second part. Assume there exists an edge e such that (s,t,e) is R-critical and v is on the detour part of some replacement path for (s,t,e). As (s,t,e) is R-critical then according to the properties of the table $\mathcal{T}_{v,t}(G)$ it follows that (i_e, d_e) is an entry of the table $\mathcal{T}_{v,t}(G)$, $d_e = d(v,t,P_{\tilde{G}}(v_{i_e}, v_{k-R}))$ and there exists a replacement path $P(v,t,e)$ such that i_e is the index of the first vertex of the detour part of $P(v,t,e)$. We claim that $i_e < i$. Assume by contradiction that $i_e \geq i$ then $P_{\tilde{G}}(s,t)[s,v_i] \circ P(v,t,e)[v_i,t]$ is also a replacement path for (s,t,e) whose common prefix with $P_{\tilde{G}}(s,t)$ contains at least R edges (as $e_i \in P^R_{\tilde{G}}(s,t)$), contradicting the assumption that (s,t,e) is R-critical. Recall that $(i_{e'}, d_{e'})$ is the entry in the table $\mathcal{T}_{v,t}(G)$ whose index $i_{e'} \leq i$ is maximal. It follows that both $i_{e'}, i_e \leq i$ and since $i_{e'}$ is the maximal index in the table $\mathcal{T}_{v,t}(G)$ such that $i_{e'} \leq i$ then $i_e \leq i_{e'}$. By the properties of the table $\mathcal{T}_{v,t}(G)$ it follows that the entries of the table are monotone and thus $d(v,t,e) = d_e \geq d_{e'} = d'(v,t,P^R_{\tilde{G}}(s,t))$. As we have already proved that $d'(v,t,P^R_{\tilde{G}}(s,t)) \geq d(v,t,e)$ then it follows that $d'(v,t,P^R_{\tilde{G}}(s,t)) = d(v,t,e)$.

The remaining of this overview is organized as follows. In Section 3.3, we describe how to compute the indices $i_{\tilde{G}}(v,t,e)$. In Section 3.4, we present more formally the construction of the tables $\mathcal{T}_{v,t}(G)$ and in Section 3.5 we show how to use these tables to efficiently compute the distances $d_G(v,t,P^R_{\tilde{G}}(s,t))$. We summarize the algorithm for handling Case 5 in Section 3.6.

3.3 The Indices $i_{\tilde{G}}(v,t,e)$

In this section, we first precisely define the set of indices $\{i_{\tilde{G}}(v,t,e)\}_{v\in B, t\in V, e\in P_{\tilde{G}}(v,t)}$ and then prove Lemma 10 by describing an algorithm that efficiently computes this set of indices.

Let $P_{\tilde{G}}(v,t) = \langle v = v_0, v_1, \ldots, v_k = t\rangle$ and let $e \in P_{\tilde{G}}(v,t)$ be an edge.

Definition 5 (The set of paths $\mathcal{P}_{\tilde{G}}(v,t,e)$). *We define the set of paths $\mathcal{P}_{\tilde{G}}(v,t,e)$ to contain all the replacement paths $P(v,t,e)$ that consists of a common prefix* $\mathrm{CommonPref}(P(v,t,e), P_{\tilde{G}}(v,t))$ *with the shortest path $P_{\tilde{G}}(v,t)$ from v to t in $\tilde{G}$, a disjoint detour* $\mathrm{Detour}(P(v,t,e), P_{\tilde{G}}(v,t))$ *and a common suffix* $\mathrm{CommonSuff}(P(v,t,e), P_{\tilde{G}}(v,t))$ *with the shortest path $P_{\tilde{G}}(v,t)$ from v to t in $\tilde{G}$.*

Definition 6 (The index $i_{\tilde{G}}(P(v,t,e))$). *Let $P(v,t,e) \in \mathcal{P}_{\tilde{G}}(v,t,e)$. We define the index $i_{\tilde{G}}(P(v,t,e))$ as the maximum index $0 \leq i \leq k$ such that $v_i \in \mathrm{CommonPref}(P(v,t,e), P_{\tilde{G}}(v,t))$ and $v_{i+1} \notin \mathrm{CommonPref}(P(v,t,e), P_{\tilde{G}}(v,t))$.*

In other words, $\mathrm{CommonPref}(P(v,t,e), P_{\tilde{G}}(v,t)) = \langle v = v_0, \ldots, v_{i_{\tilde{G}}(P(v,t,e))}\rangle$. *This also implies that $v_{i_{\tilde{G}}(P(v,t,e))}$ is the last vertex of the prefix* $\mathrm{CommonPref}(P(v,t,e), P_{\tilde{G}}(v,t))$, *and $v_{i_{\tilde{G}}(P(v,t,e))}$ is also the first vertex of the detour* $\mathrm{Detour}(P(v,t,e), P_{\tilde{G}}(v,t))$.

Definition 7 (The index $i_{\tilde{G}}(v,t,e)$). *We define the index $i_{\tilde{G}}(v,t,e)$ as an arbitrary index chosen from the group $\{i_{\tilde{G}}(P(v,t,e)) \mid P(v,t,e) \in \mathcal{P}_{\tilde{G}}(v,t,e)\}$.*

In the following section we prove that given the distances $\{d_G(v,t,e)\}_{v\in B, t\in V, e\in P_{\tilde{G}}(v,t)}$ and $\{d_G(s,v,e)\}_{s\in V, v\in B, e\in P_{\tilde{G}}(s,v)}$, one can compute efficiently the indices $\{i_{\tilde{G}}(v,t,e)\}_{v\in B, t\in V, e\in P_{\tilde{G}}(v,t)}$.

Before we explain how to compute the indices $i_{\tilde{G}}(v,t,e)$ we first note that computing the distances $\{d_G(v,t,e)\}_{v\in B, t\in V, e\in P_{\tilde{G}}(v,t)}$ and $\{d_G(s,v,e)\}_{s\in V, v\in B, e\in P_{\tilde{G}}(s,v)}$, can simply be done by invoking $\mathrm{SSRP}_G(v)$ in the graph G for all $v \in B$ and $\mathrm{SSRP}_{G^T}(v)$ in the graph G^T for all $v \in B$.

3.3.1 *An Efficient Algorithm for Computing the Indices $i_{\tilde{G}}(v,t,e)$ for Graphs with Positive Integer Edge Weights.*

In this section, we prove that given the distances $\{d_G(x,y)\}_{x,y\in V}$, $\{d_G(v,t,e)\}_{v\in B, t\in V, e\in P_{\tilde{G}}(v,t)}$ and $\{d_G(s,v,e)\}_{s\in V, v\in B, e\in P_{\tilde{G}}(s,v)}$, one can compute w.h.p. the indices $\{i_{\tilde{G}}(v,t,e)\}_{v\in B, t\in V, e\in P_{\tilde{G}}(v,t)}$ in $\widetilde{O}(Mn^3/R)$ time.

We first precisely define the input and output of this problem.

Input. The distances $\{d_G(x,y)\}_{x,y\in V}$, $\{d_G(v,t,e)\}_{v\in B, t\in V, e\in P_{\tilde{G}}(v,t)}$ and $\{d_G(s,v,e)\}_{s\in V, v\in B, e\in P_{\tilde{G}}(s,v)}$.

Output. For every $v \in B, t \in V, e \in P_{\tilde{G}}(v,t)$ such that e is not among the last R edges of $P_{\tilde{G}}(v,t)$, the algorithm computes the index $i_{\tilde{G}}(v,t,e)$ as per Definition 7. The output needs to be correct with high probability.

Remark: Given a vertex $v \in B$, the $\mathrm{SSRP}_G(v)$ algorithm of Grandoni and Vassilevska Williams [15] computes the distances $\{d(v,t,e)\}_{t\in V, e\in P_{\tilde{G}}(v,t)}$ in $\widetilde{O}(Mn^\omega)$ time. We believe that it is possible to apply small changes to the $\mathrm{SSRP}_G(v)$ algorithm of Grandoni and Vassilevska Williams [15] so that the algorithm also computes and stores the indices $i_{\tilde{G}}(v,t,e)$ using prior work. However, such an algorithm would significantly depend on prior work from multiple papers (e.g., [15], [22], [24], [4]) and will require multiple changes and the description of multiple algorithms. We therefore, in order to be more self-contained, describe in this section a self-contained reduction that does not rely on any prior work.

Computing the Indices $i_{\tilde{G}}(v,t,e)$ - a Self-Contained Reduction

Our algorithm is based on the following observation (that is also stated in the following lemma). Let $P_{\tilde{G}}(v,t) = \langle v = v_0, \ldots, v_k = t\rangle$ and let $e = (v_i, v_{i+1}) \in P_{\tilde{G}}(v,t)$ be an edge that is not among the last R edges of $P_{\tilde{G}}(v,t)$ (*i.e.*, $i < k - R$). Given a vertex $u \in B$ such that $d(v,t,e) = d(v,u,e) + d(u,t,e)$ and u is not on the subpath of $P_{\tilde{G}}(v,t)$ from v to e, one can compute the index $i_{\tilde{G}}(v,t,e)$ in $\widetilde{O}(1)$ time. In other words, if we are given another vertex $u \in B$ such that u is either on the detour part or on the suffix part of some replacement path then we can use u to compute $i_{\tilde{G}}(v,t,e)$ as follows. We can perform a binary search to find the maximum index j such that $d(v,t,e) = d(v,v_j) + d(v_j,u,e) + d(u,t,e)$. In the following lemma we show that we can set $i_{\tilde{G}}(v,t,e)$ to be j. We will later show that we can find w.h.p such a vertex u using sampling.

Lemma 11. *Let $e = (v_i, v_{i+1}) \in P_{\tilde{G}}(v,t)$. One can compute the index $i_{\tilde{G}}(v,t,e)$ in $\widetilde{O}(1)$ time, given a vertex $u \in B$ such that $d(v,t,e) = d(v,u,e) + d(u,t,e)$ and u does not belong to the subpath of $P_{\tilde{G}}(v,t)$ from v to e.*

Proof. Let $0 \le j \le i$ be the maximum index such that the following equation holds.

$$d(v,t,e) \quad = \quad d(v,v_j) + d(v_j,u,e) + d(u,t,e) \qquad (1)$$

Note that such an index j exists, since according to the assumptions of the lemma it holds that $d(v,t,e) = d(v,u,e) + d(u,t,e)$. Therefore, $d(v,t,e) = d(v,u,e) + d(u,t,e) = d(v,v) + d(v,u,e) + d(u,t,e) = d(v,v_0) + d(v_0,u,e) + d(u,t,e)$ (recall that $v_0 = v$). That is, Equation 1 holds for $j = 0$. According to Equation 1, it follows that there exists a replacement path $P(v,t,e) = \langle v_0, \ldots, v_j\rangle \circ P(v_j,u,e) \circ P(u,t,e)$ for (v,t,e). Since $0 \le j \le i$ is the maximum index such that Equation 1 holds, then $v_{j+1} \notin P(v,t,e)$. Hence, $i_{\tilde{G}}(P(v,t,e)) = j$.

Next, we explain how to find the index j in $O(\log n)$ time using a binary search. Since $0 \le j \le i$ is the maximum index such that $d(v,t,e) = d(v,v_j) + d(v_j,u,e) + d(u,t,e)$, it follows that for every $j' \le j$ the following equality holds $d(v,t,e) = d(v,v_{j'}) + d(v_{j'},u,e) + d(u,t,e)$ and for every j' such that $j < j' \le i$ the following inequality holds $d(v,t,e) > d(v,v_{j'}) + d(v_{j'},u,e) + d(u,t,e)$. Due to this monotonicity property, if we have access to the distances $d(v,v_{j'}), d(v_{j'},u,e), d(u,t,e)$ then we can use a binary search on the index j' whose boundaries are $0 \le j' \le i$ to find the index $j = i_{\tilde{G}}(v,t,e)$. Since $u \in B$ then $d(u,t,e) \in \{d_G(v,t,e)\}_{v \in B, t \in V, e \in P_{\tilde{G}}(v,t)}$ and $d(v_{j'},u,e) \in \{d_G(s,v,e)\}_{s \in V, v \in B, e \in P_{\tilde{G}}(s,v)}$, and obviously $d(v,v_{j'}) \in \{d_G(x,y)\}_{x,y \in V}$. Hence, we are given all the distances $d(v,v_{j'}), d(v_{j'},u,e)$ and $d(u,t,e)$. □

The Procedure For Computing $i_{\tilde{G}}(v,t,e)$

In this section, we describe the procedure for computing w.h.p. all the indices $i_{\tilde{G}}(v,t,e)$ for every $v \in B, t \in V, e \in P_{\tilde{G}}(v,t)$ such that e is not among the last R edges of $P_{\tilde{G}}(v,t)$ in $\widetilde{O}(Mn^3/R)$ time (w.h.p.). More precisely, we show how to find for every $v \in B, t \in V, e \in P_{\tilde{G}}(v,t)$ a vertex $u \in B$ such that $d(v,t,e) = d(v,u,e) + d(u,t,e)$ and u does not belong to the subpath of $P_{\tilde{G}}(v,t)$ from v to e, we then use Lemma 11 to compute the index $i_{\tilde{G}}(v,t,e)$ in $\widetilde{O}(1)$ time. Our algorithm for computing the indices $i_{\tilde{G}}(v,t,e)$ is as follows.

First, for every $S \in \{R \cdot 2^0, R \cdot 2^1, \ldots, R \cdot 2^{\lfloor \log \frac{n}{R} \rfloor}\}$, the algorithm samples a random subset $B_S \subseteq B$ obtained by choosing every vertex of B independently at random with probability $\frac{QRM}{S}$ for large enough constant $Q > 0$. Note that the expected size of B_S is $|B| * \frac{QRM}{S}$, and the expected size of B is $\widetilde{O}(n/R)$, thus the expected size of B_S is $\widetilde{O}(Mn/S)$.

For every $v \in B, t \in V, e \in P_{\tilde{G}}(v,t)$ such that e is not among the last R edges of $P_{\tilde{G}}(v,t)$, the algorithm finds the scale $S \in \{R \cdot 2^0, R \cdot 2^1, \ldots, R \cdot 2^{\lfloor \log \frac{n}{R} \rfloor}\}$ such that $S \le |P_{\tilde{G}}(e,t)| \le 2S$ (in other words, e is among the last $2S$ edges of $P_{\tilde{G}}(v,t)$ but not among the last S edges of $P_{\tilde{G}}(v,t)$).

The algorithm scans all the vertices of B_S until it finds a vertex $u \in B_S$ such that $d(v,t,e) = d(v,u,e) + d(u,t,e)$ and u is not contained in the subpath of $P_{\tilde{G}}(v,t)$ from v to e (we explain how to check efficiently if u is contained in the subpath of $P_{\tilde{G}}(v,t)$ from v to e as in the analysis of Lemma 14). If the algorithm does not find such a vertex $u \in B_S$ (for which $d(v,t,e) = d(v,u,e) + d(u,t,e)$ with u that is not contained in the subpath of $P_{\tilde{G}}(v,t)$ from v to e) then the algorithm fails (we prove in Lemma 13 that the algorithm does not fail w.h.p.). If the algorithm finds such a vertex $u \in B_S$, then, according to Lemma 11, given u it finds in $\widetilde{O}(1)$ time the index $i_{\tilde{G}}(v,t,e)$.

Next, we analyze the correctness and running time of the above procedure. The following auxiliary lemma follows easily as edge weights are between 1 and M.

Lemma 12. *Let $P(v,t,e) \in \mathcal{P}_{\tilde{G}}(v,t,e)$. The subpath $\mathrm{Detour}(P(v,t,e), P_{\tilde{G}}(v,t)) \circ$ $\mathrm{CommonSuff}(P(v,t,e), P_{\tilde{G}}(v,t))$ of $P(v,t,e)$ contains at least $|P_{\tilde{G}}(e,t)|/M$ edges.*

Proof. Let v_j be the first vertex of the detour $\mathrm{Detour}(P(v,t,e), P_{\tilde{G}}(v,t))$. Let $P(v_j,t,e)$ be the subpath of $P(v,t,e)$ from v_j to t. That is, $P(v_j,t,e) =$ $\mathrm{Detour}(P(v,t,e), P_{\tilde{G}}(v,t)) \circ \mathrm{CommonSuff}(P(v,t,e), P_{\tilde{G}}(v,t))$. We need to prove that $|P(v_j,t,e)| \ge |P_{\tilde{G}}(e,t)|/M$.

Since the weight of every edge is in the range $[1,M]$, it follows that $M \cdot |P(v_j,t,e)| \ge w(P(v_j,t,e)) \ge w(P_{\tilde{G}}(v_j,t)) \ge |P_{\tilde{G}}(v_j,t)| \ge |P_{\tilde{G}}(e,t)|$, where the last inequality holds as e appears after v_j along $P_{\tilde{G}}(v_j,t)$. Hence, $|P(v_j,t,e)| \ge |P_{\tilde{G}}(e,t)|/M$. □

In the following lemma, we prove the correctness of our procedure.

Lemma 13. *For every $v \in B, t \in V, e \in P_{\tilde{G}}(v,t)$ such that e is not among the last R edges of $P_{\tilde{G}}(v,t)$, the above procedure computes w.h.p. the index $i_{\tilde{G}}(v,t,e)$ as per Definition 7. In addition, w.h.p. the algorithm does not fail.*

Proof. Recall that $B \subset V$ is a random subset of V where each vertex is chosen independently uniformly at random with probability $Q_1 \ln n/R$. Furthermore, $B_S \subset B$ is a random subset of B where each vertex is chosen independently uniformly at random with probability $\frac{RMQ_2}{S}$. It follows that $B_S \subset V$ is a random subset of V where each vertex is chosen independently uniformly at random with probability $\frac{QM \ln n}{S}$ where $Q = Q_1 Q_2$.

Let $P(v,t,e) \in \mathcal{P}_{\tilde{G}}(v,t,e)$ be a replacement path for (v,t,e) whose prefix part contains the maximum number of edges. Since $e \in P_{\tilde{G}}(v,t)$ such that e is not among the last R edges of $P_{\tilde{G}}(v,t)$ then there exists a scale $S \in \{R \cdot 2^0, R \cdot 2^1, \dots, R \cdot 2^{\lfloor \log \frac{n}{R} \rfloor}\}$ such that $S \le |P_{\tilde{G}}(e,t)| \le 2S$. Then according to Lemma 12 the subpath $\mathrm{Detour}(P(v,t,e), P_{\tilde{G}}(v,t)) \circ \mathrm{CommonSuff}(P(v,t,e), P_{\tilde{G}}(v,t))$ of $P(v,t,e)$ contains at least $|P_{\tilde{G}}(e,t)|/M \ge S/M$ edges. Hence, according to Lemma 2 it holds w.h.p. that at least one of the vertices of $u \in B_S$ hits the subpath
$\mathrm{Detour}(P(v,t,e), P_{\tilde{G}}(v,t)) \circ \mathrm{CommonSuff}(P(v,t,e), P_{\tilde{G}}(v,t))$. Since u is, w.h.p., a vertex on the detour or the suffix part of $P(v,t,e)$, it follows that w.h.p. $d(v,t,e) = d(v,u,e) + d(u,t,e)$ and u is not contained in the subpath of $P_{\tilde{G}}(v,t)$ from v to e. As the algorithm scans the set of vertices in B_s, it would therefore find w.h.p. a vertex u such that $d(v,t,e) = d(v,u,e) + d(u,t,e)$ and u is not contained in the subpath of $P_{\tilde{G}}(v,t)$ from v to e. Thus, according to Lemma 11 the algorithm computes w.h.p. the index $i_{\tilde{G}}(v,t,e)$ as per Definition 7. It follows that w.h.p. the algorithm does not fail. □

Next, we analyze the running time of our procedure, which completes the proof of Lemma 10.

Lemma 14. *The above procedure takes $\widetilde{O}(Mn^3/R)$ time with high probability.*

Proof. Since $B_S \subset V$ is a random subset of V where each vertex is chosen independently uniformly at random with probability $\frac{QM \ln n}{S}$ for large enough constant $Q > 0$ (as explained in the proof of Lemma 13) then $|B_S| = \widetilde{O}(Mn/S)$ in expectation and with high probability.

In the algorithm above we mentioned that it is possible to check efficiently for a vertex $u \in B_S$ whether or not u is contained on the subpath of $P_{\tilde{G}}(v,t)$ from v to e, and that one can efficiently compute the scale $S \in \{R \cdot 2^0, R \cdot 2^1, \dots, R \cdot 2^{\lfloor \log \frac{n}{R} \rfloor}\}$ such that $S \le |P_{\tilde{G}}(e,t)| \le 2S$. There are many ways to implement it, one way to do so is as follows.

For every $v \in B, t \in V$ the algorithm creates a hash table $h_{v,t}$ as follows. Let $P_{\tilde{G}}(v,t) = \langle v = v_0, \dots, v_k = t \rangle$, the hash tables $h_{v,t}$ maps every vertex v_i to the index i (for every $0 \le i \le k$) and every edge (v_i, v_{i+1}) to the index i (for every $0 \le i < k$). In other words, for every $0 \le i \le k$ it holds that $h_{v,t}[v_i] = i$ and for every $0 \le i < k$ it holds that $h_{v,t}[(v_i, v_{i+1})] = i$. Now, u is contained in the subpath of $P_{\tilde{G}}(v,t)$ from v to e iff the hash table $h_{v,t}$ contains u and $h_{v,t}[u] \le h_{v,t}[e]$. Furthermore, the scale S can be obtained by observing that $|P_{\tilde{G}}(e,t)| = k - h_{v,t}[e]$, and then finding the scale $S \in \{R \cdot 2^0, R \cdot 2^1, \dots, R \cdot 2^{\lfloor \log \frac{n}{R} \rfloor}\}$ such that $S \le |P_{\tilde{G}}(e,t)| \le 2S$ can be done in constant time by a simple mathematical formula. For every $v \in B, t \in V$ creating the hash table $h_{v,t}$ takes linear time, so for all $v \in B, t \in V$ creating all the hash tables $\{h_{v,t}\}_{v \in B, t \in V}$ takes $\widetilde{O}(n^3/R)$ time.

The algorithm scans the set B_S to find a vertex $u \in B_S$ such that $d(v,t,e) = d(v,u,e) + d(u,t,e)$ and u is not contained in the subpath of $P_{\tilde{G}}(v,t)$ from v to e. This takes $O(|B_S|)$ time, and the size of B_S is $\widetilde{O}(Mn/S)$ with high probability. Then, given the vertex $u \in B_S$, according to Lemma 11, the algorithm finds in $\widetilde{O}(1)$ time the index $i_{\tilde{G}}(v,t,e)$.

Therefore, for each edge $e \in P_{\tilde{G}}(v,t)$ that is among the last $2S$ edges of $P_{\tilde{G}}(v,t)$ but not among the last S edges of $P_{\tilde{G}}(v,t)$, one can compute the index $i_{\tilde{G}}(v,t,e)$ in $\widetilde{O}(Mn/S)$ time. It follows that computing the indices $\{i_{\tilde{G}}(v,t,e)\}$ for all the S edges which are among the last $2S$ edges of $P_{\tilde{G}}(v,t)$ but not among the last S edges of $P_{\tilde{G}}(v,t)$ takes $\widetilde{O}(Mn)$ time. As there are only $O(\log n)$ scales S (we take S from the group $S \in \{R \cdot 2^0, R \cdot 2^1, \dots, R \cdot 2^{\lfloor \log \frac{n}{R} \rfloor}\}$), then for a fixed vertex $t \in V$, one can compute the indices $\{i_{\tilde{G}}(v,t,e)\}$ for all the edges $e \in P_{\tilde{G}}(v,t)$ in $\widetilde{O}(Mn)$ time. Therefore, one can compute the indices $\{i_{\tilde{G}}(v,t,e)\}$ for every $v \in B, t \in V, e \in P_{\tilde{G}}(v,t)$ in total $\widetilde{O}(Mn^3/R)$ time. □

3.4 Constructing the Tables $\{\mathcal{T}_{v,t}(G)\}_{v \in B, t \in V}$

Let $P_{\tilde{G}}(v,t) = \langle v = v_0, v_1, \dots, v_k = t \rangle$. The algorithm constructs the table $\mathcal{T}_{v,t}(G)$ as follows. Initialize $\mathcal{T}_{v,t}(G)$ to be an empty table. For every edge $e \in P_{\tilde{G}}(v_0, v_{k-R})$ the algorithm inserts the pair (i_e, d_e) with $i_e := i_{\tilde{G}}(v,t,e)$ and $d_e := d_G(v,t,e)$ to the table $\mathcal{T}_{v,t}(G)$ iff $d_G(v, v_{k-R}, e) + d_G(v_{k-R}, t) > d_G(v,t,e)$. Next, the algorithm sorts all the pairs of $\mathcal{T}_{v,t}(G)$ in ascending order of the first element i_e. It is easy to prove that $d_G(v, v_{k-R}, e) + d_G(v_{k-R}, t) > d_G(v,t,e)$ iff the suffix part of every replacement path for (v,t,e) contains less than R edges. The distance $d_e := d_G(v,t,e)$ was already computed, as $v \in B$ and the algorithm already computed $\mathrm{SSRP}_G(v)$ as described in Lemma 6 and its output contains w.h.p. the distance $d_G(v,t,e)$. The index $i_e := i_{\tilde{G}}(v,t,e)$ is computed using Lemma 10. We prove several properties of the table $\mathcal{T}_{v,t}(G)$.

Lemma 15. *Let $P_{\tilde{G}}(v,t) = \langle v = v_0, \dots, v_k = t \rangle$. One can construct the table $\mathcal{T}_{v,t}(G)$ in $\widetilde{O}(n)$ time such that the following properties hold (w.h.p.): (1) For every entry $(i_e, d_e) \in \mathcal{T}_{v,t}(G)$ it holds w.h.p. that $d_e = d_G(v,t,e) = d_G(v,t,P_{\tilde{G}}(v_{i_e}, v_{k-R}))$ and there exists a replacement path for (v,t,e) such that $i_e = i_{\tilde{G}}(v,t,e)$ is the index of the first vertex of its detour part. (2) For every edge $e \in P_{\tilde{G}}(v, v_{k-R})$, if there exists a vertex $s \in V$ such that (s,t,e) is R-critical and v is on the detour part of at least one replacement path of (s,t,e), then w.h.p. $(i_e, d_e) \in \mathcal{T}_{v,t}(G)$. (3) The entries of the table are monotone (i.e., if (i_e, d_e) and $(i_{e'}, d_{e'})$ are entries of the table such that $i_e \ge i_{e'}$ then $d_e \le d_{e'}$).*

Proof. Let (i_e, d_e) be an entry of the table $\mathcal{T}_{v,t}(G)$. By construction and by Lemma 10 it follows that w.h.p. there exists a replacement path $P(v,t,e)$ for (v,t,e) such that $i_e = i_{\tilde{G}}(v,t,e)$ is the index of the first vertex of its detour part.

Next, we prove that w.h.p. $d_e = d_G(v,t,e) =$
$d_G(v,t,P_{\tilde{G}}(v_{i_e}, v_{k-R}))$. Let v_j be the last vertex of the detour part of $P(v,t,e)$. As $P(v,t,e)$ is a replacement path for (v,t,e) that departs from $P_{\tilde{G}}(v,t)$ in the vertex v_{i_e} and re-enters $P_{\tilde{G}}(v,t)$ in the vertex v_j it clearly follows that $e \in P_{\tilde{G}}(v_{i_e}, v_j)$ and that $P(v,t,e)$ is disjoint from $P_{\tilde{G}}(v_{i_e}, v_j)$.

By the construction of $\mathcal{T}_{v,t}(G)$ it holds that $d(v, v_{k-R}, e)+$
$d(v_{k-R}, t) > d(v,t,e)$, therefore the suffix part of every replacement path of (v,t,e) contains less than R edges. Hence, in particular it must be that $j \ge k - R$ and thus $P_{\tilde{G}}(v_{i_e}, v_{k-R}) \subseteq P_{\tilde{G}}(v_{i_e}, v_j)$. Since $P(v,t,e)$ is disjoint from $P_{\tilde{G}}(v_{i_e}, v_j)$ and $P_{\tilde{G}}(v_{i_e}, v_{k-R}) \subseteq P_{\tilde{G}}(v_{i_e}, v_j)$

it follows that $P(v,t,e)$ is also disjoint from $P_{\tilde{G}}(v_{i_e}, v_{k-R})$. As $P(v,t,e)$ is disjoint from $P_{\tilde{G}}(v_{i_e}, v_{k-R})$ then $d_G(v,t,e) = \omega(P(v,t,e)) \geq d_G(v,t,P_{\tilde{G}}(v_{i_e}, v_{k-R}))$, where the last inequality holds as $P(v,t,e)$ is a path in $G \setminus P_{\tilde{G}}(v_{i_e}, v_{k-R})$, and $d_G(v,t,P_{\tilde{G}}(v_{i_e}, v_{k-R}))$ is the length of a shortest path in $G \setminus P_{\tilde{G}}(v_{i_e}, v_{k-R})$.

By the construction of $\mathcal{T}_{v,t}(G)$ it holds that e is not among the last R edges of $P_{\tilde{G}}(s,t)$. As $e \in P_{\tilde{G}}(v_{i_e}, v_j)$ is not among the last R edges of $P_{\tilde{G}}(s,t)$ then $e \in P_{\tilde{G}}(v_{i_e}, v_{k-R})$. Since $e \in P_{\tilde{G}}(v_{i_e}, v_{k-R})$ then $d(v,t,e) \leq d_G(v,t,P_{\tilde{G}}(v_{i_e}, v_{k-R}))$ as the distance from v to t may only increase when we remove more edges from $G \setminus \{e\}$. We proved that w.h.p. $d(v,t,e) \geq d_G(v,t,P_{\tilde{G}}(v_{i_e}, v_{k-R}))$ and $d(v,t,e) \leq d_G(v,t,P_{\tilde{G}}(v_{i_e}, v_{k-R}))$, and thus it follows that w.h.p. $d_e = d(v,t,e) = d_G(v,t,P_{\tilde{G}}(v_{i_e}, v_{k-R}))$.

Next, let $e \in P_{\tilde{G}}(v, v_{k-R})$ and assume that there exists a vertex $s \in V$ such that (s,t,e) is R-critical and v is on the detour part of a replacement path $P(s,t,e)$. We next prove that w.h.p. $(i_e, d_e) \in \mathcal{T}_{v,t}(G)$. We claim that the suffix part of every replacement path of (v,t,e) contains less than R edges. Assume by contradiction that there exists a replacement path $P'(v,t,e)$ for (v,t,e) whose suffix part contains at least R edges, then $P'(s,t,e) := P(s,t,e)[s,v] \circ P'(v,t,e)$ is a replacement path for (s,t,e) whose suffix part contains at least R edges, contradicting the assumption that (s,t,e) is R-critical. Thus, assume that the suffix part of every replacement path of (v,t,e) contains less than R edges. It is easy to prove that in this case $d_G(v, v_{k-R}, e) + d_G(v_{k-R}, t) > d(v,t,e)$ and thus w.h.p. the algorithm added the entry (i_e, d_e) into the table $\mathcal{T}_{v,t}(G)$.

Finally, we prove the following monotonicity property on the entries of the table $\mathcal{T}_{v,t}(G)$. Let $e = (v_i, v_{i+1})$ and $e' = (v_j, v_{j+1})$ $(0 \leq i, j < k - R)$ be two edges such that both (i_e, d_e) and $(i_{e'}, d_{e'})$ are entries of the table $\mathcal{T}_{v,t}(G)$. We assume that $i_e \geq i_{e'}$ and prove that $d_e \geq d_{e'}$. We already proved in this lemma that $d_e = d_G(v,t,e) = d_G(v,t,P_{\tilde{G}}(v_{i_e}, v_{k-R}))$ and $d_{e'} = d_G(v,t,e') = d_G(v,t,P_{\tilde{G}}(v_{i_{e'}}, v_{k-R}))$. As we assume that $i_e \geq i_{e'}$ then $P_{\tilde{G}}(v_{i_e}, v_{k-R}) \subset P_{\tilde{G}}(v_{i_{e'}}, v_{k-R})$ and thus $d_e = d_G(v,t,P_{\tilde{G}}(v_{i_e}, v_{k-R})) \leq d_G(v,t,P_{\tilde{G}}(v_{i_{e'}}, v_{k-R})) = d_{e'}$ as the distance from v to t may only increase when we remove more edges from $G \setminus P_{\tilde{G}}(v_{i_e}, v_{k-R})$.

□

3.5 Computing the Distances $\{d'(v,t,P^R_{\tilde{G}}(s,t))\}_{v \in B, s,t \in V}$

Note that due to uniqueness of shortest paths in $\tilde{G}$ we have that the paths $P_{\tilde{G}}(v,t)$ and $P_{\tilde{G}}(s,t)$ have some common suffix (and are disjoint up to this common suffix). In order to estimate the distance $d_G(v,t,P^R_{\tilde{G}}(s,t))$ the algorithm does the following. Let $P_{\tilde{G}}(v,t) = \langle v_0, ..., v_k \rangle$ and let $e_i = (v_i, v_{i+1})$ be the first common edge of $P_{\tilde{G}}(v,t)$ and $P_{\tilde{G}}(s,t)$ (see figure 2). In the remaining of this section assume that e_i is not among the first or last R edges of $P_{\tilde{G}}(s,t)$ (we handle these end cases in the full version, but for simplicity we omit it from the overview). If there is no entry $(i_{e'}, d_{e'})$ in the table $\mathcal{T}_{v,t}(G)$ such that $i_{e'} \leq i$ then return $d'(v,t,P^R_{\tilde{G}}(s,t)) = \infty$. Otherwise, let $(i_{e'}, d_{e'})$ be the maximum entry in the table such that $i_{e'} \leq i$ (as the table is sorted then finding this pair can easily be done using a binary search). Return $d'(v,t,P^R_{\tilde{G}}(s,t)) = d_{e'}$.

Next, we sketch the proof of correctness. We prove the following lemma for the more general and difficult case that $e, e_i \in P^R_{\tilde{G}}(s,t) \cap P_{\tilde{G}}(v,t)$, in the full version we also handle the easier cases that $e_i \notin P^R_{\tilde{G}}(s,t)$ and/or $e \notin P_{\tilde{G}}(v,t)$. We claim that when $e, e_i \in P^R_{\tilde{G}}(s,t) \cap P_{\tilde{G}}(v,t)$ then the last R edges of $P_{\tilde{G}}(s,t)$ and $P_{\tilde{G}}(v,t)$ are the same. To see that, note that by the uniqueness of shortest paths in $\tilde{G}$, it follows that $P_{\tilde{G}}(s,t) \cap P_{\tilde{G}}(v,t)$ is a common suffix of $P_{\tilde{G}}(s,t)$ and $P_{\tilde{G}}(v,t)$ that contains e, e_i and as $e_i \in P^R_{\tilde{G}}(s,t)$ thus the suffix CommonSuff$(P_{\tilde{G}}(v,t), P_{\tilde{G}}(s,t))$ contains more than R edges.

LEMMA 16. *Let $e \in P^R_{\tilde{G}}(s,t) \cap P_{\tilde{G}}(v,t)$ and let $e_i = (v_i, v_{i+1})$ be the first edge of $P_{\tilde{G}}(v,t) \cap P_{\tilde{G}}(s,t)$ and assume $e_i \in P^R_{\tilde{G}}(s,t)$. The following conditions hold: (1) $d'(v,t,P^R_{\tilde{G}}(s,t)) \geq d(v,t,e)$. (2) If (s,t,e) is R-critical and v is on the detour part of some replacement path for (s,t,e) then $d'(v,t,P^R_{\tilde{G}}(s,t)) = d(v,t,e)$.*

PROOF SKETCH. We first prove that $d'(v,t,P^R_{\tilde{G}}(s,t)) \geq d(v,t,e)$. As $e, e_i \in P^R_{\tilde{G}}(s,t) \cap P_{\tilde{G}}(v,t)$ and $e_i = (v_i, v_{i+1})$ is the first edge of $P_{\tilde{G}}(v,t) \cap P_{\tilde{G}}(s,t)$ it follows that e appears between v_i and v_{k-R} along $P_{\tilde{G}}(v,t)$. If there is no entry $(i_{e'}, d_{e'})$ in the table $\mathcal{T}_{v,t}(G)$ such that $i_{e'} \leq i$ then $d'(v,t,P^R_{\tilde{G}}(s,t)) = \infty$ and thus it trivially holds that $d'(v,t,P^R_{\tilde{G}}(s,t)) \geq d(v,t,e)$. Assume that there exists at least one entry $(i_{e'}, d_{e'})$ in the table $\mathcal{T}_{v,t}(G)$ such that $i_{e'} \leq i$ and let $(i_{e'}, d_{e'})$ be the entry in the table whose index $i_{e'} \leq i$ is maximal. By construction we have that $d'(v,t,P^R_{\tilde{G}}(s,t)) = d_{e'}$ and according to Lemma 15 it holds w.h.p. that $d_{e'} = d_G(v,t,P_{\tilde{G}}(v_{i_{e'}}, v_{k-R})) \geq d(v,t,e)$ where the last inequality holds since e appears between v_i and v_{k-R} along $P_{\tilde{G}}(v,t)$ and $i_{e'} \leq i$. Thus, $d'(v,t,P^R_{\tilde{G}}(s,t)) \geq d(v,t,e)$.

Next, we assume that (s,t,e) is R-critical and v is on the detour part of at least one replacement path $P(s,t,e)$ and prove that $d'(v,t,P^R_{\tilde{G}}(s,t)) = d(v,t,e)$. According to Lemma 15 it holds w.h.p. that $(i_e, d_e) \in \mathcal{T}_{v,t}(G), d_e = d(v,t,e) = d_G(v,t,P_{\tilde{G}}(v_{i_e}, v_{k-R}))$ and there exists a replacement path $P(v,t,e) \in \mathcal{P}(v,t,e)$ such that i_e is the index of the first vertex of the detour part of $P(v,t,e)$. We claim that $i_e < i$. Assume by contradiction that $i_e \geq i$ then $P_{\tilde{G}}(s,t)[s,v_i] \circ P(v,t,e)[v_i,t]$ is also a replacement path for (s,t,e) whose common prefix with $P_{\tilde{G}}(s,t)$ contains at least R edges (as $e_i \in P^R_{\tilde{G}}(s,t)$), contradicting the assumption that (s,t,e) is R-critical.

Recall that $(i_{e'}, d_{e'})$ is the entry in the table $\mathcal{T}_{v,t}(G)$ whose index $i_{e'} \leq i$ is maximal. It follows that both $i_{e'}, i_e \leq i$ and since $i_{e'}$ is the maximal index in the table $\mathcal{T}_{v,t}(G)$ such that $i_{e'} \leq i$ then $i_e \leq i_{e'}$. According to Lemma 15 the entries of the table are monotone and thus $d(v,t,e) = d_e \geq d_{e'} = d'(v,t,P^R_{\tilde{G}}(s,t))$. As we have already proved that $d'(v,t,P^R_{\tilde{G}}(s,t)) \geq d(v,t,e)$ then it follows that w.h.p. $d'(v,t,P^R_{\tilde{G}}(s,t)) = d(v,t,e)$.

□

3.6 The Algorithm for Handling Case 5

To sum up, we briefly describe the algorithm for handling case 5. The goal is to compute (and store), for every $s,t \in V$ such that $d^{\leq R}(s,t) > d(s,t)$ the distance $d_5(s,t)$ which is an estimation of the distance $d_G(s,t,P^R_{\tilde{G}}(s,t))$.

- For every $v \in B$ compute $\mathrm{SSRP}_G(v)$ according to Lemma 6 in the graph G and $\mathrm{SSRP}_{G^T}(v)$ in the graph G^T with reverse edge directions in $\widetilde{O}(\frac{n}{R} \cdot \mathrm{SSRP}(M,n)) = \widetilde{O}(\frac{n}{R} \cdot Mn^{\omega})$ time.
- Compute the set of indices $\{i_{\tilde{G}}(v,t,e)\}_{v \in B, t \in V, e \in P_{\tilde{G}}(v,t)}$ as in Lemma 10 in $\widetilde{O}(Mn^3/R)$ time.
- For every $v \in B, t \in V$, let $P_{\tilde{G}}(v,t) = \langle v = v_0, v_1, \ldots, v_k = t \rangle$, the algorithm constructs the table $\mathcal{T}_{v,t}(G)$ in $\widetilde{O}(n)$ time as follows. Initialize $\mathcal{T}_{v,t}(G)$ to be an empty table. For every edge $e \in P_{\tilde{G}}(v, v_{k-R})$ with $d_G(v, v_{k-R}, e) + d_G(v_{k-R}, t) > d_G(v,t,e)$ the algorithm inserts the pair (i_e, d_e) such that w.h.p. $d_e := d_G(v,t,e)$ as computed by the $\mathrm{SSRP}_G(v)$ algorithm, and the index $i_e := i_{\tilde{G}}(v,t,e)$ as computed by the algorithm described in Lemma 10. Next, the algorithm sorts all the pairs of $\mathcal{T}_{v,t}(G)$ in ascending order of the first element i_e.
- For every $s,t \in V$ such that $d^{\leq R}(s,t) > d(s,t)$ and for every $v \in B$ the algorithm computes in $\widetilde{O}(1)$ time an estimate $d'(v,t,P^R_{\tilde{G}}(s,t))$ of the distance $d_G(v,t,P^R_{\tilde{G}}(s,t))$ as follows. Using a binary search, find the minimum index $0 \leq i \leq k$ such that v_i is the first common vertex of $P_{\tilde{G}}(s,t)$ and $P_{\tilde{G}}(v,t)$. To see that, as shortest paths in $\tilde{G}$ are unique, then i can be found using a binary search on the vertices of $P_{\tilde{G}}(v,t) = \langle v_0, \ldots, v_k \rangle$ by searching for minimum index i such that v_i also belongs to the path $P_{\tilde{G}}(s,t)$, and in each iteration of the binary search the algorithm tests whether or not a node v_i belongs to the path $P_{\tilde{G}}(s,t)$ in constant time using the LCA data-structures (see more details in the full version). Next, using another binary search, find the entry $(i_{e'}, d_{e'})$ in the table $\mathcal{T}_{v,t}(G)$ such that $i_{e'} \leq i$ is maximal (as the table is sorted then finding this pair can easily be done using a binary search). Set $d'(v,t,P^R_{\tilde{G}}(s,t)) = d_{e'}$.

  We run the above procedure also in the graph G^T with reversed edge directions which computes the distances $d'_{G^T}(v,s,P^R_{\tilde{G}^T}(t,s))$. For clarity, denote by $d'(s,\mathbf{v},P^R_{\tilde{G}}(s,t)) := d'_{G^T}(v,s,P^R_{\tilde{G}^T}(t,s))$.
- Finally, for every $s,t \in V$ such that $d^{\leq R}(s,t) > d(s,t)$, the algorithm computes and stores $d_5(s,t)$ as follows $d_5(s,t) = \min_{v \in B}\{d'(s,\mathbf{v},P^R_{\tilde{G}}(s,t)) + d'(v,t,P^R_{\tilde{G}}(s,t))\}$.

We summarize the correctness of the algorithm in the following lemma.

Lemma 17. *The following conditions hold: (1) For every edge $e \in P^R_{\tilde{G}}(s,t)$ it holds that $d_5(s,t) \geq d(s,t,e)$. (2) If there exists an edge e such that (s,t,e) is R-critical (i.e., (s,t,e) belongs to Case 5) then it holds that $d_5(s,t,e) = d_G(s,t,e)$ with high probability.*

Proof. To see the first requirement, let $e \in P^R_{\tilde{G}}(s,t)$, by reversing edge directions it holds that $e^T \in P^R_{G^T}(t,s)$. For every $v \in B$ it holds by Lemma 16 on the graph G^T that $d'_{G^T}(\mathbf{v},s,P^R_{G^T(t,s)}) \geq d_{G^T}(\mathbf{v},s,e^T) = d_G(s,v,e)$ and according to Lemma 16 on the graph G it holds that $d'(\mathbf{v},t,P^R_{\tilde{G}}(s,t)) \geq d(\mathbf{v},t,e)$. Hence, $d_5(s,t) = \min_{v \in B}\{d'(s,\mathbf{v},P^R_{\tilde{G}}(s,t)) + d'(\mathbf{v},t,P^R_{G}(s,t))\} \geq \min_{v \in B}\{d(s,v,e) + d(v,t,e)\} \geq d(s,t,e)$ where the last inequality holds by the triangle inequality in the graph $G \setminus \{e\}$.

We are left showing the second part. Let e be an edge such that (s,t,e) is R-critical. According to the definition of an R-critical query the detour part of any replacement path $P(s,t,e)$ contains at least R edges, and according to Lemma 2 it holds with high probability that at least one of the vertices $v \in B$ belongs to the detour of $P(s,t,e)$. Furthermore, according to Lemma 9 it holds that $d(s,t,e) = d(s,t,P^R_{\tilde{G}}(s,t))$. Hence, it holds with high probability that $d(s,t,e) = d(s,t,P^R_{\tilde{G}}(s,t)) = d(s,v,P^R_{G}(s,t)) + d(v,t,P^R_{\tilde{G}}(s,t)) = d'(\mathbf{v},s,P^R_{\tilde{G}}(s,t)) + d'(\mathbf{v},t,P^R_{\tilde{G}}(s,t)) \geq d_5(s,t)$ where the last equality holds due to Lemma 16. We get that $d_5(s,t) \geq d(s,t,e)$ and w.h.p. $d(s,t,e) \geq d_5(s,t)$ so w.h.p. $d_5(s,t) = d(s,t,e)$. □

4 HANDLING NEGATIVE WEIGHTS - AN OVERVIEW

There are two main technical issues with handling negative weights. The first is that the best known SSRP algorithm for negative weights is substantially slower than the one for positive weights. The second issue is that computing the index $i_{\tilde{G}}(v,t,e)$ as described in Lemma 10 heavily relies on the assumption that edge weights are positive integers in the range $[1,M]$, so that a path of length ℓ contains at least ℓ/M edges. This is not true in the presence of non-positive weights.

We redefine the cases that the algorithm handles slightly differently (mainly in order to handle the issue that we now need $O(\log n)$ different scales for computing the SSRP algorithms). The main difference in the algorithm is in handling Case 5 and in particular in constructing the tables $\mathcal{T}_{v,t}(G)$.

We next describe the different cases when handling negative weights. We then highlight the main differences in the construction of the tables in Case 5. We refer the reader to the full version for a complete description of our DSO for negative weights.

Cases 0-3 are essentially the same as in the DSO for positive weights. The one slight difference is that in the computation of the partial shortest paths trees $\{\tilde{T}_s\}_{s \in V}$ our algorithm invokes Dijkstra, which does not work well in graphs with negative edge weights. To overcome this issue we simply use the known method of feasible price function [16] (see more details in the full version).

Case 0: There exists at least one shortest path from s to t in G that contains at most R edges. In other words, this case happens iff $d^{\leq R}(s,t) = d(s,t)$.

Case 1: $e \notin P_{\tilde{G}}(s,t)$.

Case 2: e is among the first or last R edges of the unique shortest path $P_{\tilde{G}}(s,t)$.

Case 3: There exists a replacement path from s to t in $G \setminus \{e\}$ that contains at most $3R$ edges.

Case 4': Let $\alpha = \alpha(s,t,e) \in \mathcal{S} = \{R \cdot 2^0, R \cdot 2^1, \ldots, R \cdot 2^{\lfloor \log \frac{n}{R} \rfloor}\}$ be the minimum integer such that $|P_{\tilde{G}}(s,e)| \leq 2\alpha(s,t,e)$ and $\beta = \beta(s,t,e) \in \mathcal{S}$ be the minimum integer such that $|P_{\tilde{G}}(e,t))| \leq 2\beta(s,t,e)$. We say that (s,t,e) belongs to Case 4' if one of the following conditions hold.

(a) There exists $P(s,t,e) \in \mathcal{P}_{\tilde{G}}(s,t,e)$ such that $|\mathrm{CommonPref}(P(s,t,e), P_{\tilde{G}}(s,t))| \geq \alpha$; Or,

(b) There exists $P(s,t,e) \in \mathcal{P}_{\tilde{G}}(s,t,e)$ such that $|\mathrm{CommonSuff}(P(s,t,e), P_{\tilde{G}}(s,t))| \geq \beta$; Or,

(c) There exists $P(s,t,e) \in \mathcal{P}_{\tilde{G}}(s,t,e)$ such that
($|\text{Detour}(P(s,t,e), P_{\tilde{G}}(s,t))| \leq |\text{CommonPref}(P(s,t,e), P_{\tilde{G}}(s,t))|$
or
$|\text{Detour}(P(s,t,e), P_{\tilde{G}}(s,t))| \leq |\text{CommonSuff}(P(s,t,e), P_{\tilde{G}}(s,t))|$).

Case 5': The complement of the previous cases. The triple (s,t,e) belongs to Case 5' if it does not belong to any of the cases 0-3 and 4'. We refer to Case 5' as the negative-critical case, and a triple (s,t,e) that belongs to Case 5' is called a negative-critical query.

Handling Case 4'. To handle Case 4', recall that in the case of positive weights, Case 4 was defined as the case that either the prefix or the suffix parts of at least one replacement path contains at least R edges. Let v_s be the first vertex of B along $P_{\tilde{G}}(v,t)$ and let v_t be the last vertex of B along $P_{\tilde{G}}(v,t)$, then in the case of positive weights the algorithm computes $d_4(s,t,e) = \min\{d(s,v_s) + \text{SSRP}_G(v_s,t,e), \text{SSRP}_{G^T}(v_t,s,e^T) + d(v_t,t)\}$.

However, in the presence of negative edge weights we do not have an estimation of $\text{SSRP}_G(v,t,e)$ and $\text{SSRP}_{G^T}(s,v,e^T)$ for every $v \in B$. In order to handle this issue we use $O(\log n)$ different scales for computing the SSRP algorithms. For every $S \in \mathcal{S}$, let $B_S \subset V$ be a random set of vertices obtained by choosing every vertex independently uniformly at random with probability $\frac{Q \ln n}{S}$ for large enough constant $Q > 0$. Let $S \in \mathcal{S}, v \in B_S$ we use the $\text{SSRP}^{3S}_{G}(v)$ algorithm described by Grandoni and Vassilevska Williams in Lemma 8 in [15] that computes correctly w.h.p. all the replacement path distances of paths from v in G with at most $3S$ vertices. For increasing $S \in \mathcal{S}$, each execution of the modified $\text{SSRP}^{3S}_{G}(v)$ algorithm becomes more expensive, but this is compensated by the smaller number of executions (*i.e.*, $\widetilde{O}(n/S)$).

Furthermore, the algorithm finds and stores the vertex $\text{first}^{S}_{\tilde{G}}(s,t) \in B_S$ that is the first vertex of B_S along $P_{\tilde{G}}(s,t)$ when traversed from s to t, and the vertex $\text{last}^{S}_{\tilde{G}}(s,t) \in B_S$ that is the last vertex of B_S along $P_{\tilde{G}}(s,t)$ when traversed from s to t (or sets them to null if no such vertices exist). In the full version we describe how to efficiently compute the vertices $\text{first}^{S}_{\tilde{G}}(s,t)$ and $\text{last}^{S}_{\tilde{G}}(s,t)$. Given a query (s,t,e) such that $d^{\leq R}(s,t) > d(s,t)$, the algorithm computes $d_{4'}(s,t,e) = \min_{S \in \mathcal{S}, v \in \{\text{first}^S(s,t,e), \text{last}^S(s,t,e)\}}\{\text{SSRP}^{3S}_{G^T}(v,s,e) + \text{SSRP}^{3S}_{G}(v,t,e)\}$.

Handling Case 5'. The main technical complicated part of handling Case 5' is in the construction of the tables $\mathcal{T}_{v,t}(G)$ in the presence of negative edge weights. In contrary to the case of positive weights where the $\text{SSRP}_G(v)$ algorithm computes replacement paths on arbitrary number of edges, in the case of negative weights the $\text{SSRP}^{3S}_{G}(v)$ algorithm is guaranteed to compute correctly the length of replacement paths on at most $3S$ edges (w.h.p.). We now associate each table with two additional parameters β and S such that $\beta, S \in \mathcal{S}$, where β and S represent that we are looking for replacement paths on roughly S edges whose suffix part contains less than β edges. For every $\beta, S \in \mathcal{S}$ and $v \in B_S$ the algorithm constructs a table $\mathcal{T}^{\beta,S}_{v,t}(G)$ whose properties are somewhat similar to the properties of the table $\mathcal{T}_{v,t}(G)$ in the case of positive weights, but the construction algorithm is different than in the case of positive weights.

Ideally, for every $e \in P_{\tilde{G}}(v,t)$ such that the suffix part of every replacement path for (v,t,e) contains less than β edges, we would like to compute the entry (i_e, d_e) such that $d_e = d(v,t,e)$ and $i_e = i_{\tilde{G}}(v,t,e)$ is the index of the first vertex of the detour part of some replacement path for (v,t,e). However, the computation of the index $i_{\tilde{G}}(v,t,e)$ as done in the case of positive weights in Lemma 10 heavily relies on the fact that a path of length ℓ contains at least ℓ/M edges. This is not true in the presence of non-positive weights. To overcome this difficulty, we describe a different algorithm that computes $i_e := \rho_{\tilde{G}}(v,t,e)$ where $\rho_{\tilde{G}}(v,t,e)$ is defined as the maximum number of edges in a prefix of a replacement path for (v,t,e). In Section 5.5 of the full version we describe an algorithm that computes $\rho_{\tilde{G}}(v,t,e)$ correctly w.h.p. in $\widetilde{O}(n/R)$ time as stated in the following lemma.

Lemma 18. *[Proof in the full version] Let $S \in \mathcal{S}, v \in B_S, t \in V, e \in P_{\tilde{G}}(v,t)$, there exists an algorithm that computes in $\widetilde{O}(n/R)$ time an integer $i_e := \rho_{\tilde{G}}(v,t,e)$ such that w.h.p. i_e the maximum number of edges in a prefix of a replacement path for (v,t,e).*

Even though we are looking for replacement paths $P(v,t,e)$ on at most S edges, it could be that the original shortest path $P_{\tilde{G}}(v,t)$ contains $\Omega(n)$ edges, and so computing the entry (i_e, d_e) for every edge $e \in P_{\tilde{G}}(v,t)$ in $\widetilde{O}(n/R)$ time is too expensive. Let $P^{\leq S}(v,t)$ be the shortest v-to-t path on at most S edges, note that for every $e \notin P^{\leq S}(v,t)$ it holds that $P^{\leq S}(v,t)$ is a shortest path on at most S edges from v to t that avoids e. As we are looking for replacement paths on at most S edges then we can return $P^{\leq S}(v,t)$ in case $e \notin P^{\leq S}(v,t)$. We thus only need to compute the entries (i_e, d_e) for the $O(S)$ edges in $P_{\tilde{G}}(v,t) \cap P^{\leq S}(v,t)$. However, for large S it is not known how to efficiently compute the paths $\{P^{\leq S}(v,t)\}_{S \in \mathcal{S}, v \in B_S, t \in V}$. Note that it is crucial for our algorithm that for each path $P^{\leq S}(v,t)$ not only that its length is at most the length of the shortest v-to-t path on at most S edges, but also that $P^{\leq S}(v,t)$ contains only $O(S)$ edges. In the full version we describe an algorithm that, loosely speaking, efficiently computes APSP with $O(S)$ edges. We compute a set of paths $\{P^{\widetilde{\leq S}}_{G}(v,t)\}_{v \in B_S, t \in V}$ such that $w(P^{\widetilde{\leq S}}_{G}(v,t)) \leq d^{\leq S}(v,t)$ and $|P^{\widetilde{\leq S}}_{G}(v,t)| \leq 3S$. In other words, we compute a path that contains at most $3S$ edges, whose length is shorter or has equal length as the shortest v-to-t path on at most S edges. The notation $P^{\widetilde{\leq S}}_{G}(v,t)$ stands for a shortest v-to-t path with approximately S edges.

Lemma 19. *[Proof in the full version] Let G be a weighted graph with integer edge weights in the range $[-M, M]$, let $S > 0$ be an integer parameter and let $B_S \subseteq V$ be a set of $\widetilde{O}(n/S)$ vertices. One can compute in $\widetilde{O}(Mn^{\omega+1/2})$ time a set of distances $\{d^{\widetilde{\leq S}}_{G}(v,t)\}_{v \in B_S, t \in V}$ and a set of paths $\{P^{\widetilde{\leq S}}_{G}(v,t)\}_{v \in B_S, t \in V}$ such that $d^{\widetilde{\leq S}}_{G}(v,t) := w(P^{\widetilde{\leq S}}_{G}(v,t)) \leq d^{\leq S}(v,t)$ and $|P^{\widetilde{\leq S}}_{G}(v,t)| \leq 3S$.*

5 FURTHER IMPROVING THE PREPROCESSING TIME

The runtime of the construction algorithm of the *DSO* described above (including computing *APSP*, $APSP^{\leq R}$, the partial shortest paths trees $\{\tilde{T}_s\}_{s \in V}$, $\{LCA(\tilde{T}_s)\}_{s \in V}$, the hash tables h_0 and h_2, and the data-structures DSO_3, DSO_4 and DSO_5) takes $\widetilde{O}(n/R \cdot Mn^{\omega} + RM^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}} + Mn^{\omega} \cdot (X^{4-\omega} + \sqrt{n}) + n^3 \frac{R}{X} + Mn^3/R)$ time. The exponent of n is minimized when $R = M^{\frac{4-\omega}{9-2\omega}} n^{\frac{\omega^2-6\omega+7}{2\omega-9}}$ and $X = M^{\frac{-1}{9-2\omega}} n^{\frac{4-\omega}{9-2\omega}}$, then the preprocessing time is $\widetilde{O}(M^{\frac{5-w}{9-2\omega}} n^{\frac{16+\omega-\omega^2}{9-2\omega}})$, which is subcubic for $\omega \in [2, 2.373]$. For $\omega = 2.373$ we get subcubic preprocessing $\widetilde{O}(M^{0.62}n^{2.9953})$ time.

In this section, we outline how to improve the preprocessing time of our DSO to $\widetilde{O}(Mn^{\omega+1/2})$ for $\omega \in [2.35, 2.373]$. The main idea is to improve the running time of computing the $\mathrm{SSRP}^{3S}_{G}(v)$ algorithm for all $S \in \mathcal{S}, v \in B_S$. More precisely, instead of computing independently every instance of the $\mathrm{SSRP}^{3S}_{G}(v)$ algorithm we develop a new algorithm referred to as *BATCH-SSRP* algorithm, that computes together and more efficiently the set of distances $\{\mathrm{SSRP}^{3S}_{G}(v,t,e)\}_{S\in\mathcal{S}, v\in B_S, t\in V, e\in P_{\tilde{G}}(s,t)}$

In Section 4, for a particular $S \in \mathcal{S}$ and $v \in B_S$ the algorithm computed $\mathrm{SSRP}^{3S}_{G}(v)$ using the algorithm described by Grandoni and Vassilevska Williams in Lemma 8 in [15] in $\widetilde{O}(Mn^{\omega}+SM^{\frac{1}{4-\omega}}n^{1+\frac{1}{4-\omega}})$ time, therefore, for all $S \in \mathcal{S}$ ($|\mathcal{S}| = O(\log n)$) and $v \in B_S$ ($|B_S| = \widetilde{O}(n/S)$ w.h.p.) computing $\mathrm{SSRP}^{3S}_{G}(v)$ takes w.h.p. $\widetilde{O}(n/R \cdot Mn^{\omega} + M^{\frac{1}{4-\omega}}n^{2+\frac{1}{4-\omega}})$ time. In this section we improve the runtime of this part to $\widetilde{O}(Mn^{\omega+1/2} + n^3/R)$.

Let $\mathcal{S}_1 = \{S \mid S \in \mathcal{S} \text{ AND } S \leq \sqrt{n}\}$ and let $\mathcal{S}_2 = \{S \mid S \in \mathcal{S} \text{ AND } S > \sqrt{n}\}$. Note that for every $S \in \mathcal{S}_2$ there is a small number of vertices in B_S (at most $n/S = \widetilde{O}(\sqrt{n})$) and therefore it would be efficient enough to invoke the $\mathrm{SSRP}^{3S}_{G}(v)$ algorithm for every $v \in B_S$ as in Lemma 8 in [15] in $\widetilde{O}(|B_S| \cdot (Mn^{\omega}+M^{\frac{1}{4-\omega}}S \cdot n^{1+\frac{1}{4-\omega}})) = \widetilde{O}(Mn^{\omega+1/2} + M^{\frac{1}{4-\omega}}n^{2+\frac{1}{4-\omega}}) = \widetilde{O}(Mn^{\omega+1/2})$ time, where the first equality holds since $|B_S| = \widetilde{O}(n/S) = \widetilde{O}(\sqrt{n})$ (as $S \in \mathcal{S}_2$).

However, the size of B_S for $S \in \mathcal{S}_1$ is too large and therefore to compute the values $\{\mathrm{SSRP}^{S}_{G}(v,t,e)\}_{S\in\mathcal{S}_1, v\in B_S, t\in V, e\in P_{\tilde{G}}(s,t)}$ we take a different approach. For every $S \in \mathcal{S}_1$ the algorithm samples $X_S = S \cdot C \cdot \log n$ random graphs $\{G_1, \ldots, G_{X_S}\}$, where $X_S = S \cdot C \log n$ (for large enough constant $C > 0$) and where each G_i is obtained from G by independently removing each edge with probability $\frac{1}{S}$ (similar to the random graphs obtained in [21]). Then, the algorithm computes the STSP algorithm of Grandoni and Vassilevska Williams (see Theorem 4 from [15], given two subsets of nodes $\tilde{S}, \tilde{T} \subseteq V$, the STSP algorithm computes w.h.p. all the distances between pairs $(s,t) \in \tilde{S} \times \tilde{T}$ in $\widetilde{O}(Mn^{\omega} + |\tilde{S}| \cdot |\tilde{T}| \cdot (Mn)^{\frac{1}{4-\omega}})$ time) with $\tilde{S} = B_S$ and $\tilde{T} = V$ in every graph G_i. *I.e.*, the algorithm computes the distances $d_{G_i}(v,t)$ for every $(v,t) \in B_S \times V, 1 \leq i \leq X_s$ in $\widetilde{O}(X_S \cdot (Mn^{\omega} + n^2/S \cdot (Mn)^{\frac{1}{4-\omega}})) = \widetilde{O}(SMn^{\omega} + n^2 \cdot (Mn)^{\frac{1}{4-\omega}}) = \widetilde{O}(Mn^{\omega+1/2} + n^2 \cdot (Mn)^{\frac{1}{4-\omega}})$ time (where the last equality holds as $S \in \mathcal{S}_1$ and thus $S \leq \sqrt{n}$). For every edge $e \in E$ the algorithm computes the set of indices $F^S_e \subseteq \{1, \ldots, X_S\}$ such that $\{G_i \mid i \in F^S_e\}$ are all the graphs that do not contain e. Weimann and Yuster [21] proved that $|F^S_e| = O(\log n)$. It is easy to compute F^S_e for every e in $\widetilde{O}(X_S|E|) = \widetilde{O}(S|E|)$ time which is dominated by $\widetilde{O}(S \cdot Mn^{\omega})$ time. For every $S \in \mathcal{S}_1, v \in B_S, t \in V, e \in P_{\tilde{G}}(v,t)$ the algorithm computes $\mathrm{SSRP}^{3S}_{G}(v,t,e) = \min\{d_{G_i}(v,t) \mid i \in F^S_e\}$ in $\widetilde{O}(n^3/S) \leq \widetilde{O}(n^3/R)$ time (here we used the fact that $S \geq R$ and w.h.p. $|B_S| = \widetilde{O}(n/S)$ and $|F^S_e| = \widetilde{O}(1)$). Summing the running time for every $S \in \mathcal{S}_1 \cup \mathcal{S}_2$ we get that the total processing time for computing $\{\mathrm{SSRP}^{S}_{G}(v,t,e)\}_{S\in\mathcal{S}, v\in B_S, t\in V, e\in P_{\tilde{G}}(s,t)}$ is $\widetilde{O}(Mn^{\omega+1/2} + M^{\frac{1}{4-\omega}}n^{2+\frac{1}{4-\omega}} + n^3/R) = \widetilde{O}(Mn^{\omega+1/2} + n^3/R)$.

It remains to analyze the total preprocessing time of our DSO. Following this improvement, the preprocessing time of the DSO reduces to $\widetilde{O}(Mn^{\omega+1/2} + RM^{\frac{1}{4-\omega}}n^{2+\frac{1}{4-\omega}} + Mn^{\omega} \cdot (X^{4-\omega} + \sqrt{n}) + n^3\frac{R}{X} + Mn^3/R)$. For $\omega \in [2.35, 2.373]$, the above running time is minimized when $R = \widetilde{O}(n^{2.5-\omega})$, $X = \widetilde{O}(n^{5-2\omega})$ and then the total preprocessing time is $\widetilde{O}(Mn^{\omega+1/2})$. For $\omega = 2.373$ the preprocessing time is $\widetilde{O}(Mn^{2.873})$.

6 SPACE ANALYSIS

In this section, we prove that the space of our DSO is $\widetilde{O}(n^{2.5})$ (where the size is measured in words, assuming any relevant distance can be stored in one word).

The following data-structures that are stored as part of our DSO require $\widetilde{O}(n^2R)$ space. The distances $\{d(s,t), d^{\leq R}(s,t)\}_{s,t\in V}$ are stored using $O(n^2)$ space, the hash table h_0 requires $O(n^2R)$ space (as it contains $O(n^2R)$ triples (s,t,e) such that $s,t \in V$, and e is one of the edges of a shortest path $P^{\leq R}(s,t)$ that contains at most R edges), the hash table h_2 requires $O(n^2R)$ space (as it contains $O(n^2R)$ triples (s,t,e) such that $s,t \in V$ and e is among the first or last R edges of $P_{\tilde{G}}(s,t)$), the data structures $\{LCA(\tilde{T}_s)\}_{s\in V}$ require $O(n^2)$ space, the data structure DSO_3 uses $\widetilde{O}(n^2R)$ space (as DSO_3 is constructed by sampling $\widetilde{O}(R)$ random graphs $\{G_i\}$, where each G_i is obtained from G by independently removing each edge with probability $\frac{1}{R}$, and for every graph G_i we store the result of running APSP in G_i using $O(n^2)$ space per graph G_i), the vertices $\mathrm{first}^S_{G}(s,t)$ and $\mathrm{last}^S_{G}(s,t)$ for all $s,t \in V, S \in \mathcal{S}$ are stored using $\widetilde{O}(n^2)$ space, and the data structure DSO_5 requires $\widetilde{O}(n^2)$ space (as for every pair $s,t \in V$ we store $\widetilde{O}(1)$ distances).

In addition, the algorithm stores for every $S \in \mathcal{S}$ and $v \in B_S$ the output of $\mathrm{SSRP}^{3S}_{G}(v)$ and $\mathrm{SSRP}^{3S}_{G^T}(v)$ algorithms in the graph G and in the graph G^T, which requires $\widetilde{O}(n^3/R)$ space. The $\widetilde{O}(n^3/R)$ space is too large, and in the following we mitigate it to $\widetilde{O}(n^{2.5})$ space.

The idea is not to explicitly store all the $\{\mathrm{SSRP}^{S}_{G}(v,t,e)\}_{S\in\mathcal{S}, v\in B_S, t\in V, e\in P_{\tilde{G}}(s,t)}$ values, but instead use a variant of the data-structure described in Section 5 to construct a data-structure that requires $\widetilde{O}(n^{2.5})$ space such that given $S \in \mathcal{S}, v \in B_S, t \in V, e \in P_{\tilde{G}}(s,t)$ one can compute $\mathrm{SSRP}^{S}_{G}(v,t,e)$ in $\widetilde{O}(1)$ time using this data structure.

As in Section 5, for every $S \in \mathcal{S}_2$ the preprocessing algorithm simply invokes the $\mathrm{SSRP}^{S}_{G}(v)$ algorithm for every $v \in B_S$ using the algorithm described by Grandoni and Vassilevska Williams in Lemma 8 in [15] and stores the $O(n^2)$ computed distances for every invocation of the $\mathrm{SSRP}^{S}_{G}(v)$ algorithm, this requires $\widetilde{O}(|B_S| \cdot n^2) = \widetilde{O}(n^{2.5})$ space, where the last equality holds since $|B_S| = \widetilde{O}(n/S) = \widetilde{O}(\sqrt{n})$ (as $S \in \mathcal{S}_2$).

For every $S \in \mathcal{S}_1$, the preprocessing algorithm computes the STSP algorithm using the algorithm of Grandoni and Vassilevska Williams (see Theorem 4 in [15]) with $\tilde{S} = B_S$ and $\tilde{T} = V$ in every graph $G_i \in \{G_1, \ldots, G_{X_S}\}$, (where the graphs $\{G_1, \ldots, G_{X_S}\}$ are defined as in Section 5), *i.e.*, computes and stores the distances $d_{G_i}(v,t)$ for every $(v,t) \in B_S \times V, 1 \leq i \leq X_S$ using $\widetilde{O}(X_S \cdot |B_S| \cdot |V|) = \widetilde{O}(S \cdot n/S \cdot n) = \widetilde{O}(n^2)$ space. For every edge $e \in E$ the algorithm also computes and stores the set of indices $F^S_e \subseteq \{1, \ldots, X_S\}$ such that $\{G_i \mid i \in F_e^S\}$ are all the graphs that do not contain e using $\widetilde{O}(m)$ space (as $|F_e^S| = \widetilde{O}(1)$).

During query time, given $S \in \mathcal{S}_1, v \in B_S, t \in V, e \in P_{\tilde{G}}(v,t)$ the algorithm computes $\mathrm{SSRP}_G^S(v,t,e) = \min\{d_{G_i}(v,t) \mid i \in F_e^S\}$ in $\widetilde{O}(1)$ time (here we used the fact that $|F_e^S| = \widetilde{O}(1)$). Given $S \in \mathcal{S}_2, v \in B_S, t \in V, e \in P_{\tilde{G}}(v,t)$ the algorithm simply restores the precomputed distance $\mathrm{SSRP}_G^S(v,t,e)$.

We get that the total space used by the DSO is $\widetilde{O}(n^2 R + n^{2.5})$. Since we have $R = \widetilde{O}(\sqrt{n})$ it follows that our DSO requires $\widetilde{O}(n^{2.5})$ space.

REFERENCES

[1] [n.d.].

[2] Ittai Abraham, Shiri Chechik, and Cyril Gavoille. 2012. Fully Dynamic Approximate Distance Oracles for Planar Graphs via Forbidden-set Distance Labels. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing (STOC)*. 1199–1218. https://doi.org/10.1145/2213977.2214084

[3] Ittai Abraham, Shiri Chechik, Cyril Gavoille, and David Peleg. 2010. Forbidden-set distance labels for graphs of bounded doubling dimension. In *PODC*. 192–200.

[4] Noga Alon, Zvi Galil, Oded Margalit, and Moni Naor. 1992. Witnesses for Boolean Matrix Multiplication and for Shortest Paths. In *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992*. 417–426. https://doi.org/10.1109/SFCS.1992.267748

[5] Surender Baswana, Utkarsh Lath, and Anuradha S. Mehta. 2012. Single Source Distance Oracle for Planar Digraphs Avoiding a Failed Node or Link. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 223–232. http://dl.acm.org/citation.cfm?id=2095116.2095136

[6] Michael A Bender and Martin Farach-Colton. 2000. The LCA problem revisited. In *Latin American Symposium on Theoretical Informatics*. Springer, 88–94.

[7] Aaron Bernstein and David Karger. 2008. Improved Distance Sensitivity Oracles via Random Sampling. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 34–43. http://dl.acm.org/citation.cfm?id=1347082.1347087

[8] Panagiotis Charalampopoulos, Shay Mozes, and Benjamin Tebeka. 2019. Exact Distance Oracles for Planar Graphs with Failing Vertices. (2019). To appear.

[9] Shiri Chechik, Sarel Cohen, Amos Fiat, and Haim Kaplan. 2017. $(1+\epsilon)$-Approximate f-sensitive Distance Oracles. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms* (Barcelona, Spain) *(SODA '17)*. 1479–1496. http://dl.acm.org/citation.cfm?id=3039686.3039782

[10] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. 2012. f-Sensitivity Distance Oracles and Routing Schemes. *Algorithmica* 63, 4 (2012), 861–882. https://doi.org/10.1007/s00453-011-9543-0

[11] Bruno Courcelle and Andrew Twigg. 2007. Compact Forbidden-Set Routing. In *STACS*. 37–48. https://doi.org/10.1007/978-3-540-70918-3_4

[12] Bruno Courcelle and Andrew Twigg. 2010. Constrained-Path Labellings on Graphs of Bounded Clique-Width. *Theory Comput. Syst.* 47, 2 (2010), 531–567. http://springerlink.metapress.com/content/b3268gtk313180q0/

[13] Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. 2008. Oracles for Distances Avoiding a Failed Node or Link. *SIAM J. Comput.* 37, 5 (Jan. 2008), 1299–1318. https://doi.org/10.1137/S0097539705429847

[14] Ran Duan and Seth Pettie. 2009. Dual-failure Distance and Connectivity Oracles. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 506–515. http://dl.acm.org/citation.cfm?id=1496770.1496826

[15] Fabrizio Grandoni and Virginia Vassilevska Williams. 2012. Improved Distance Sensitivity Oracles via Fast Single-Source Replacement Paths. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*. 748–757. https://doi.org/10.1109/FOCS.2012.17

[16] Donald B. Johnson. 1977. Efficient Algorithms for Shortest Paths in Sparse Networks. *J. ACM* 24, 1 (Jan. 1977), 1–13. https://doi.org/10.1145/321992.321993

[17] Neelesh Khanna and Surender Baswana. 2010. 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*. 513–524. https://doi.org/10.4230/LIPIcs.STACS.2010.2481

[18] François Le Gall. 2014. Powers of Tensors and Fast Matrix Multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation* (Kobe, Japan) *(ISSAC '14)*. ACM, New York, NY, USA, 296–303.

[19] Avi Shoshan and Uri Zwick. 1999. All Pairs Shortest Paths in Undirected Graphs with Integer Weights. In *In IEEE Symposium on Foundations of Computer Science*. 605–614.

[20] Jan van den Brand and Thatchaphol Saranurak. 2019. Sensitive Distance and Reachability Oracles for Large Batch Updates. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*. 424–435. https://doi.org/10.1109/FOCS.2019.00034

[21] Oren Weimann and Raphael Yuster. 2013. Replacement Paths and Distance Sensitivity Oracles via Fast Matrix Multiplication. *ACM Trans. Algorithms* 9, 2 (2013), 14. https://doi.org/10.1145/2438645.2438646

[22] Virginia Vassilevska Williams. 2011. Faster Replacement Paths. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*. 1337–1346. https://doi.org/10.1137/1.9781611973082.102

[23] Virginia Vassilevska Williams. 2012. Multiplying Matrices Faster Than Coppersmith-winograd. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing* (New York, New York, USA) *(STOC '12)*. ACM, New York, NY, USA, 887–898. https://doi.org/10.1145/2213977.2214056

[24] Uri Zwick. 2002. All Pairs Shortest Paths Using Bridging Sets and Rectangular Matrix Multiplication. *J. ACM* 49, 3 (2002), 289–317.