A Nearly Linear Time Construction of Approximate Single-Source Distance Sensitivity Oracles

Kaito Harada
Osaka University, Suita, Japan

Naoki Kitamura
Osaka University, Suita, Japan

Taisuke Izumi
Osaka University, Suita, Japan

Toshimitsu Masuzawa
Osaka University, Suita, Japan

Abstract

An *α-approximate vertex fault-tolerant distance sensitivity oracle* (α-*VSDO*) for a weighted input graph $G = (V, E, w)$ and a source vertex $s \in V$ is the data structure answering an α-approximate distance from s to t in $G - x$ for any given query $(x, t) \in V \times V$. It is a data structure version of the so-called single-source replacement path problem (SSRP). In this paper, we present a new *nearly linear-time* algorithm of constructing a $(1 + \epsilon)$-VSDO for any directed input graph with polynomially bounded integer edge weights. More precisely, the presented oracle attains $\tilde{O}(m \log(nW)/\epsilon + n \log^2(nW)/\epsilon^2)$* construction time, $\tilde{O}(n \log(nW)/\epsilon)$ size†, and $\tilde{O}(1/\epsilon)$ query time, where n is the number of vertices, m is the number of edges, and W is the maximum edge weight. These bounds are all optimal up to polylogarithmic factors. To the best of our knowledge, this is the first non-trivial algorithm for SSRP/VSDO beating $\tilde{O}(mn)$ computation time for directed graphs with general edge weight functions, and also the first nearly linear-time construction breaking approximation factor 3. Such a construction has been unknown even for undirected and unweighted graphs. In addition, our result implies that the known conditional lower bounds for the exact SSRP computation does not apply to the case of approximation.

2012 ACM Subject Classification Theory of computation → Shortest paths; Theory of computation → Data structures design and analysis

Keywords and phrases data structure, distance sensitivity oracle, replacement path problem, graph algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.65

Related Version *Full Version*: https://arxiv.org/abs/2401.01103

Funding This work was supported by MEXT/JSPS KAKENHI Grant Number 22K21277, 23K16838, 23K24825, and 23H04385.

1 Introduction

1.1 Background and Our Result

The fault of links and vertices is ubiquitous in real-world networks. In fault-prone networks, it is important to develop an algorithm that quickly recompute the desired solution for a given fault pattern. For example, suppose that a client wants to send information to all guests in the network, but the shortest paths can be disconnected by faults. Then, it needs

[*] The $\tilde{O}(\cdot)$ notation omits polylogarithmic factors, i.e., $\tilde{O}(f(n)) = O(f(n)\text{polylog}(n))$.

[†] Throughout this paper, size is measured by the number of words. One word is $O(\log n)$ bits.

32nd Annual European Symposium on Algorithms (ESA 2024).
Editors: Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman; Article No. 65; pp. 65:1–65:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to find the shortest paths avoiding all failed entities, so-called *replacement paths*. In this paper, we consider the *Single Source Replacement Path Problem* (*SSRP*) for directed graph $G = (V(G), E(G))$ with positive edge weights, which requires us to find the single-source shortest paths for all possible single vertex-fault patterns. That is, its output is the distances from a given source vertex s to any vertex $t \in V(G)$ in graphs $G - x$ for every $x \in V(G)$. This problem is known as one of the fundamental problems, not only in the context of fault-tolerance, but also in the auction theory [21, 27]. While SSRP is also considered in the edge-fault case, it is easily reduced to the vertex-fault case of SSRP. Hence this paper only focuses on vertex faults.

A trivial algorithm for SSRP is to solve the single-source shortest path problem on $G - x$ for each failed vertex $x \in V(G) \setminus \{s\}$. The running time of this algorithm is $\tilde{O}(mn)$, where n is the number of vertices and m is the number of edges. While it is an intriguing question whether non-trivial speedup of computing SSRP is possible or not, a variety of conditional lower bounds has been presented (see Table 3). Focusing on directed graphs with arbitrary positive edge weights, $\Omega(mn)$ time lower bound in the path comparison model [23] has been proved by Hershberger, Suri, and Bhosle [22]. A similar fine-grained complexity barrier is also provided by Chechik and Cohen [10], which exhibits $\tilde{\Omega}(mn^{1-\delta})$ lower bound for any small constant $\delta > 0$ under the assumption that there exists no boolean matrix multiplication algorithm running in $O(mn^{1-\delta})$ time for $n \times n$ matrices with a total number of m 1s. Due to these results, the quest for much faster algorithms essentially requires some relaxation on the problem setting. The research line considering undirected and/or unweighted graphs recently yields much progress. For directed and unweighted graphs, a randomized SSRP algorithm running in $\tilde{O}(m\sqrt{n} + n^2)$ time has been shown [12], which is also proved conditionally tight under some complexity-theoretic hypotheses.

In this paper, we aim to circumvent the barriers above by admitting approximation. While such an approximation approach is known and succeeded for the s-t replacement path problem (RP) [3], there have been no results so far for SSRP. A trivial bottleneck of SSRP is $O(n^2)$ term to output the solution containing the distances for all possible pairs of a failed vertex and a target vertex. This bottleneck slightly spoils the challenge to approximate SSRP algorithms of $o(m\sqrt{n}) + O(n^2)$ running time because it benefits only when $m = \omega(n^{3/2})$ holds. Hence this paper focuses on the fast construction of the *α-approximate vertex fault-tolerant distance sensitivity oracle* (*α-VSDO*), which is the compact (i.e. $o(n^2)$ size) data structure answering an α-approximate distance from s to t in $G - x$ for any query (x, t). This is naturally regarded as the data structure version of SSRP, and due to its compactness, the time for outputting the solution does not matter anymore. There are several results on the construction of α-DSOs. However, in the case of $\alpha = (1 + \epsilon)$, no construction algorithm running faster than $O(m\sqrt{n})$ time is invented, even for undirected and unweighted graphs (see Tables 1 and 2). The results explicitly faster than $O(m\sqrt{n})$ time are by Baswana and Khanna [2] and Bilò, Guala, Leucci, and Proietti [8], which attain only 3-approximation for undirected graphs. Our main contribution is to solve this open problem positively and almost completely. The main theorem is stated as follows:

▶ **Theorem 1.** *Given any directed graph G with edge weights in range $[1, W]$, a source vertex $s \in V(G)$, and a constant $\epsilon \in (0, 1]$, there exists a deterministic algorithm of constructing a $(1 + \epsilon)$-VSDO of size $O(\epsilon^{-1} n \log^3 n \cdot \log(nW))$. The construction time and the query processing time are respectively $O(\epsilon^{-1} \log^4 n \cdot \log(nW)(m + n\epsilon^{-1} \cdot \log^3 n \cdot \log(nW)))$ and $O(\log^2 n \cdot \log(\epsilon^{-1} \log(nW)))$.*

For polynomially-bounded edge weights and $\epsilon = \Omega(1/\text{polylog}(n))$, one oracle attains $\tilde{O}(m)$ construction time, $\tilde{O}(n)$ size, and $\tilde{O}(1)$ query processing time, which are optimal up to polylogarithmic factors. It also deduces an algorithm for $(1 + \epsilon)$-approximate SSRP running in $\tilde{O}(m + n^2)$ trivially. To the best of our knowledge, this is the first non-trivial result for SSRP/VSDO beating $O(mn)$ computation time for directed graphs with polynomially-bounded positive edge weights, and also the first nearly linear-time construction of breaking approximation factor 3. Such a construction has been unknown even for undirected and unweighted graphs.

We emphasize that nearly linear time construction of approximate DSOs inherently faces the challenge that oracles must be built *without explicitly computing SSRP*. If poly(n) construction time is admitted, one can adopt the two-phase approach which computes the SSRP at first, and then compresses the computation result into a small-size oracle. In fact, this approach is adopted by most of known $(1 + \epsilon)$-approximate constructions. It is, however, impossible in considering $\tilde{O}(m)$-time construction. The high-level structure of our construction algorithm is the combination of the divide-and-conquer approach used in Grandoni and Vassilevska Williams [18] and Chechik and Magen[12], and the *progressive Dijkstra* algorithm by Bernstein [3] originally designed for solving the approximate s-t replacement path problem. The crux of our result is to demonstrate that the progressive Dijkstra provides much useful information for computing approximate SSRP beyond RP, by carefully installing it into the approach of [12, 18]. In addition, as a by-product, we refine the original progressive Dijkstra algorithm into a simpler and easy-to-follow form, which is of independent interest and potentially useful for its future applications.

1.2 Related Work

The known results directly related to our algorithm are summarized in Tables 1, 2, and 3. We explain some supplementary remark on these tables. The BCHR20 construction [1] is only the $(1 + \epsilon)$-approximation result which does not rely on the explicit SSRP computation. However, the construction is based on the information hard to compute in $\tilde{O}(m)$ time (the information $HD_\alpha(v)$ presented at Section 3 in [1]). While the BCHR20 does not state the construction time explicitly, it requires $O(mn)$ time following the authors' observation. The SSRP algorithm by Chechik and Magen [12] is referred to as the one for unweighted graphs, but it can also cover bounded rational edge weights in the range $[1, c]$ with a constant $c > 1$. It is not easy to transform this algorithm into an approximate SSRP handling general edge weights by scaling approaches, because edge weights are *lower bounded* by one. Some of the results in Table 1 [4, 14, 20] actually provide the more stronger oracles which support all-pair or multiple-source queries.

There are a variety of topics closely related to SSRP/DSO. A *f-fault-tolerant (approximate) shortest path tree* is a subgraph of the input graph G which contains a (approximate) shortest path tree in $G - F$ for any faulty edge set F of size at most f [9, 28, 29]. Any 1-fault tolerant shortest path tree can be seen as a graph representation of the output of (approximate) SSRP, and its construction is closely related to the construction of DSOs. In fact, some of the results in Table 1 also include the construction of fault-tolerant shortest path trees [1, 8].

The oracles and algorithms covering multiple failures are also investigated [5, 11, 13, 16, 17, 31]. The most general structure is the *f-sensitivity all-pairs oracle*, which returns an exact or approximate length of the shortest replacement path avoiding a given faulty edge set F such that $|F| \leq f$ holds. Currently, most of oracles covering general f faults assume undirected graphs, and the construction time and the oracle size are essentially more expensive than single-source 1-sensitivity oracles.

Table 1 Known Results for EDSO/VDSO. The column “problem” describes the target problem of each result, where the first additional character V/E represents the fault model (vertex fault/edge fault). The column “input” describes the type of graphs each result covers. In the notation X/Y, X is either D (directed) or U (undirected), and the Y is either U (unweighted), + (positive edge weight), or ± (arbitrary edge weight). The column “apx.” represents approximation factors. The symbol L is the shorthand of $\epsilon^{-1}\log(nW)$. The dagger mark † implies some additional features not described in the table, which is explained in Section 1.2.

ref	problem	input	apx.	size	construction	query
DT02[14]	EDSO†	D/+	1	$O(n^2\log n)$	$\tilde{O}(mn^2)$	$\tilde{O}(1)$
BK09[4]	EDSO†	D/+	1	$O(n^2)$	$\tilde{O}(mn)$	$O(1)$
BK13[2]	VDSO	U/+	3	$O(n\log n)$	$\tilde{O}(m)$	$O(1)$
BK13[2]	VDSO	U/U	$1+\epsilon$	$O(\frac{n}{\epsilon^3}+n\log n)$	$O(m\sqrt{\frac{n}{\epsilon}})$	$O(1)$
BGLP16[7]	EDSO	U/+	2	$O(n)$	$\tilde{O}(mn)$	$O(1)$
BGLP16[7]	EDSO	U/+	$1+\epsilon$	$O(\frac{n}{\epsilon}\log\frac{1}{\epsilon})$	$\tilde{O}(mn)$	$O(\log n\cdot\frac{1}{\epsilon}\log\frac{1}{\epsilon})$
GS18[20]	ESDO†	U/U	1	$\tilde{O}(n^{3/2})$	$\tilde{O}(mn)$	$\tilde{O}(1)$
BGLP18[8]	EDSO	U/+	3	$O(n)$	$\tilde{O}(m)$	$O(1)$
BGLP18[8]	EDSO	U/U	$1+\epsilon$	$O(\frac{n}{\epsilon^3})$	$O(m\sqrt{\frac{n}{\epsilon}})$	$O(1)$
BCHR20[1]	VDSO	D/+	$1+\epsilon$	$O(nL)$	$O(mn)^{\dagger}$	$O(\log L)$
BCFS21[6]	ESDO	U/U	1	$\tilde{O}(n^{3/2})$	$\tilde{O}(m\sqrt{n}+n^2)$	$\tilde{O}(1)$
DG22[15]	ESDO	U/U	1	$\tilde{O}(n^{3/2})$	$\tilde{O}(m\sqrt{n})$	$\tilde{O}(1)$
This paper	VDSO	D/+	$1+\epsilon$	$O(nL\log^3 n)$	$\tilde{O}(mL+nL^2)$	$O(\log^2 n\cdot\log L)$

Table 2 Known Upper Bounds for RP and SSRP, where ω is the matrix multiplication exponent, and $\alpha(n,m)$ is the inverse of the Ackermann function.

ref	problem	input	apx.	time
MMG89 [24]	ERP	U/+	1	$O(m+n\log n)$
NPW03 [26]	VRP	U/+	1	$O(m+n\log n)$
NPW03 [25]	ERP	U/+	1	$O(m\alpha(n,m))$
Ber10 [3]	ERP	D/+	$1+\epsilon$	$\tilde{O}(m\log L)$
Vas11 [32]	ERP	D/±	1	$\tilde{O}(Wn^{\omega})$
GV12 [18]	ESSRP	D/±	1	$\tilde{O}(W^{\frac{1}{4-\omega}}n^{2+\frac{1}{4-\omega}})$
GV12 [18]	ESSRP	D/+	1	$\tilde{O}(Wn^{\omega})$
RZ12 [30]	ERP	D/U	1	$O(m\sqrt{n})$
CC19 [10]	ESSRP	U/U	1	$\tilde{O}(m\sqrt{n}+n^2)$
CM20 [12]	ESSRP	D/U†	1	$\tilde{O}(m\sqrt{n}+n^2)$
GPWX21 [19]	ESSRP	D/±	1	$\tilde{O}(M^{\frac{5}{17-4\omega}}n^{\frac{36-7\omega}{17-4\omega}})$

2 Preliminary

Let $V(H)$ and $E(H)$ denote the vertex and edge set of a directed graph H respectively, and $w_H(u,v)$ or $w_H(e)$ denote the weight of edge $e=(u,v)$ in H. Let G be the input directed graph of n vertices and m edges with integer edge weights within $[1,W]$. We assume edge weights are polynomially bounded, i.e., $W=\text{poly}(n)$. Since we only consider approximate shortest paths, the assumption of integer edge weights is not essential, because real weights can be rounded with an appropriate precision.

For a vertex set $U\subseteq V(H)$ of a graph H, $H-U$ denotes the graph obtained by removing the vertices in U and the edges incident to them from H. Similarly, for an edge set $U\subseteq E(H)$, $H-U$ denotes the graph obtained by removing the edges in U. We often use the abbreviated notation $H-u$ when U consists of a single element u. Let $N_H(u)$ be the set of the neighbors of $u\in V(H)$ in H. For a vertex subset $U\subseteq V(H)$, let $\Psi(U)$ be the set of the edges incident to a vertex $u\in U$. Note that $\Psi(U)$ contains the edges whose endpoints are both in U. Given a set X of vertices or edges of H, we define $H[X]$ as the subgraph induced by X.

Table 3 Known conditional lower bounds for RP and SSRP. All results apply to the case of exact computation.

ref	problem	input	time	model/hypothesis
HSB07 [22]	ERP	D/+	$\Omega(m\sqrt{n})$	Path comparison model
HSB07 [22]	ESSRP	D/+	$\Omega(mn)$	Path comparison model
VW18 [33]	ERP	D/±	$\Omega(n^{3-\delta})\log W$	no subcubic APSP
VW18 [33]	ERP	D/U	$\Omega(mn^{1/2-\delta})$	no subcubic BMM
CC19 [10]	ESSRP	U/U	$\Omega(mn^{1/2-\delta}+n^2)$	no $O(mn^{1-\delta'})$ BMM
CC19 [10]	Comb. E-SSRP	U/+	$\Omega(mn^{1-\delta})$	no Comb. subcubic APSP
CC20 [12]	ESSRP	U/+	$\Omega(mn^{1/2-\delta}+n^2)$	no subcubic APSP

A sequence of vertices $A = \langle a_0, a_1, \ldots, a_{k-1}\rangle$ such that $(a_i, a_{i+1}) \in E(H)$ $(0 \le i < k-1)$ is called a path from a_0 to a_{k-1} in H. We use the terminology "path" for a simple path (i.e., a_i for $0 \le i \le k-1$ are all different). For simplicity, we often deal with a path A as the path subgraph corresponding to A. We define $A[i, j]$ as its sub-path $\langle a_i, a_{i+1}, \ldots, a_j\rangle$, and $w(A)$ as the weighted length of A. The (directed) distance $\mathsf{dist}_H(u, v)$ from u to v in H is the length of the u-v shortest path in H. For any set Σ of paths, we define $\mathsf{minL}(\Sigma)$ as $\mathsf{minL}(\Sigma) = \min\{w(Q) \mid Q \in \Sigma\}$. We also define $\mathsf{MinP}(\Sigma)$ as the path $Q \in \Sigma$ of length $\mathsf{minL}(\Sigma)$. If two or more paths have length $\mathsf{minL}(\Sigma)$, an arbitrary one of them is chosen as a canonical path.

The problem considered in this paper is defined as follows.

▶ **Definition 2** ($(1+\epsilon)$-VSDO). *A $(1+\epsilon)$-VSDO for a directed graph G with positive weights, a single source $s \in V(G)$, and a positive constant $\epsilon \in (0, 1]$ is the data structure supporting query $\textsf{dso-query}(v_f, x)$ for any failed vertex $v_f \in V(G) \setminus \{s\}$ and destination $t \in V(G)$ which returns a value satisfying $\mathsf{dist}_{G-v_f}(s, t) \le \textsf{dso-query}(v_f, x) \le (1+\epsilon)\cdot \mathsf{dist}_{G-v_f}(s, t)$.*

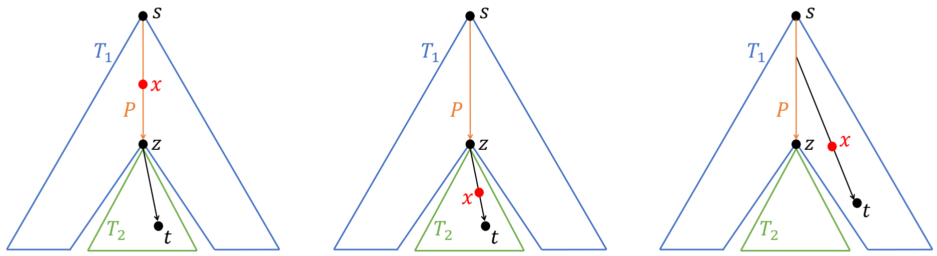
3 Technical Outline

In our construction, we recursively construct the oracle based on the *centroid bipartition*, which is the approach also adopted in [18] and [12]. Given a tree T' of n vertices, a *centroid* $z \in V(T')$ of T' is the vertex in $V(T')$ such that any connected component of $T - z$ contains at most $n/2$ vertices. Using the centroid z, one can split T' into two edge-disjoint connected subtrees T'_1 and T'_2 of T' such that $E(T'_1) \cup E(T'_2) = E(T')$, $V(T'_1) \cap V(T'_2) = \{z\}$, and $\frac{n}{3} \le |V(T'_1)|, |V(T'_2)| \le \frac{2n}{3}$, which we call the *centroid bipartition* of T'. If T' is a rooted tree, we treat both T'_1 and T'_2 also as rooted trees. The split tree containing the original root s is referred to as T'_1, whose root is s, and the other one is referred to as T'_2, whose root is z'. It is easy to find the centroid bipartition T'_1 and T'_2 from T' in $O(n)$ time. We also denote the s-z path in T' by $P_{T'}$.

Given the input graph G and a source vertex s, our algorithm first constructs a shortest path tree T rooted by s, and its centroid bipartition T_1 and T_2. The whole oracle for G and s consists of three sub-oracles. Given a query $(x, f) \in (V(G) \setminus \{s\}) \times V(G)$, they respectively handle the three cases below (see Fig. 1).

1. $x \in V(P_T)$ and $t \in V(T_2) \setminus \{z\}$
2. $x \in V(T_2) \setminus \{z\}$ and $t \in V(T_2) \setminus \{z\}$
3. $x \in V(T_1)$ and $t \in V(T_1)$

Notice that, in the first case, we use $x \in V(P_T)$ instead of $x \in V(T_1)$; this is only the non-trivial situation because $\mathsf{dist}_G(s, t) = \mathsf{dist}_{G-x}(s, t)$ obviously holds if $x \notin V(P_T)$. The



(a) Case1 **(b)** Case2 **(c)** Case3

Figure 1 Three cases in the oracle construction.

sub-oracles for the second and third cases are recursively constructed for the graphs G_2 and G_1 obtained by slightly modifying $G[V(T_2)]$ and $G[V(T_1)]$. Since the modification does not increase the sizes of G_1 and G_2 so much from $G[V(T_1)]$ and $G[V(T_2)]$, one can guarantee that the recursion depth is logarithmic. Hence the total construction time is easily bounded by $\tilde{O}(m)$ if the times of constructing the first-case sub-oracle, G_1, and G_2 are all $\tilde{O}(m)$.

3.1 Sub-oracle for the First Case

The precise goal for the first case is to develop a *Q-faulty* $(1+\epsilon_1)$*-VSDO*, which is a $(1+\epsilon_1)$-approximate VSDO only supporting the faults on a given path Q, which is formally defined as follows:

▶ **Definition 3** (Q-faulty $(1+\epsilon_1)$-VSDO)**.** *A Q-faulty $(1+\epsilon_1)$-VSDO for a directed graph G with positive weights, a source vertex $s \in V(G)$, and a path Q from s is the data structure supporting the query* $\mathsf{pf\text{-}query}_Q(x,t)$ *for any $x \in V(Q)$ and $t \in V(G)$ which returns a value satisfying* $\mathsf{dist}_{G-x}(s,t) \leq \mathsf{pf\text{-}query}_Q(x,t) \leq (1+\epsilon_1)\cdot \mathsf{dist}_{G-x}(s,t)$.

Obviously, the P_T-faulty $(1+\epsilon_1)$-VSDO for G is the sub-oracle covering the first case, where we need to set $\epsilon_1 = \epsilon/(3\log n)$ to deal with the accumulation of approximation error in the recursive construction (this point is explained later in more details). Let $P_T = \langle v_0, v_1, \ldots, v_{p-1}\rangle$ ($s = v_0, z = v_{p-1}$), and v_f be the alias of the faulty vertex $x \in V(P_T)$. The construction of the P_T-faulty $(1+\epsilon_1)$-VSDO follows a generalized and simplified version of the approximate s-t replacement path algorithm by Bernstein [3]. We first introduce two types for s-t paths in $G - v_f$ (see also Fig. 2).

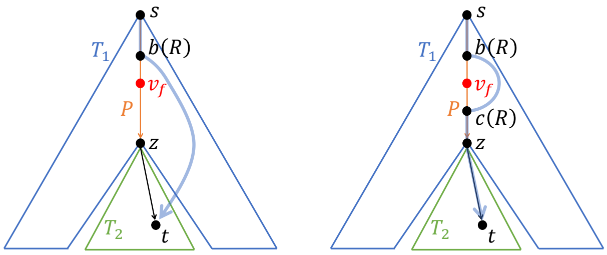
▶ **Definition 4** (Jumping and departing paths)**.** *For any failed vertex $v_f \in V(P_T)\setminus\{s\}$ and destination $t \in V(G)$, a s-t path R in $G - v_f$ is called a* departing path *avoiding v_f if it satisfies the following two conditions (C1) and (C2):*

- *(C1) The vertex subset $V(R) \cap V(P_T[0, f-1])$ induces a prefix of R.*
- *(C2) $(V(R)\setminus\{t\}) \cap V(P_T[f+1, p-1]) = \emptyset$.*

A s-t path R in $G - v_f$ is called a jumping path *avoiding v_f if it satisfies the condition (C1) and (C3) below:*

- *(C3) $(V(R)\setminus\{t\}) \cap V(P_T[f+1, p-1]) \neq \emptyset$.*

We call the last vertex of the prefix induced by $V(R) \cap V(P_T[0, f-1])$ the *branching vertex* of R, which we refer to as $b(R)$. In addition, for any jumping path R avoiding v_f, we define $c(R)$ as the first vertex in $V(R) \cap V(P_T[f+1, p-1])$ (with respect to the order of R), which we refer to as the *coalescing vertex* of R. Given any failed vertex $v_f \in V(P_T)\setminus\{s\}$ and destination $t \in V(G)$, $\mathsf{ddist}_{G-v_f}(s,t)$ and $\mathsf{jdist}_{G-v_f}(s,t)$ respectively denote the lengths of the shortest departing and jumping paths from s to t avoiding v_f. Since any path satisfies either (C2) and (C3), and one can assume that any shortest s-t path in $G - v_f$ satisfies (C1) without



(a) departing path **(b)** jumping path

Figure 2 Examples of departing and jumping paths. $b(R)$ represents the branching vertex on each path, and $c(R)$ represents a coalescing vertex in (b).

loss of generality, $\mathsf{dist}_{G-v_f}(s,t) = \min\{\mathsf{ddist}_{G-v_f}(s,t), \mathsf{jdist}_{G-v_f}(s,t)\}$ holds. Hence it suffices to construct two data structures approximately answering the values of $\mathsf{ddist}_{G-v_f}(s,t)$ and $\mathsf{jdist}_{G-v_f}(s,t)$ for any $v_f \in V(P_T)$ and $t \in V(T_2)$ respectively. For $\mathsf{ddist}_{G-v_f}(s,t)$, we realize it as an independent sub-sub-oracle:

▶ **Definition 5** (DP-Oracle). *A* Q-faulty $(1+\epsilon_1)$-DPO (DP-oracle) *for a graph G, a source vertex $s \in V(G)$, and a path Q from s is the data structure supporting the query* dp-query(x,t) *for any $x \in V(Q)$ and $t \in V(G)$ which returns a value satisfying* $\mathsf{ddist}_{G-x}(s,t) \leq$ *dp-query*$(x,t) \leq (1+\epsilon_1) \cdot \mathsf{ddist}_{G-x}(s,t)$*.*

It should be noted that our definition of P_T-faulty $(1+\epsilon_1)$-DPOs supports queries for all $t \in V(G)$, not limited to $V(T_2)$. This property is not necessary for addressing the first case, but used in other cases.

The key technical ingredient of implementing DP-oracles is the *progressive Dijkstra* algorithm presented in [3]. We first introduce several notations and terminologies for explaining it: since $V(R) \cap V(P_T[0, f-1]) = V(R) \cap V(P_T)$ necessarily holds for any departing path avoiding v_f, the branching vertex $b(R)$ is the last vertex of the path induced by $V(R) \cap V(P_T)$, i.e., $b(R)$ is determined only by R, independently of v_f. It allows us to treat departing paths without association of avoiding vertex v_f. That is, the sentence "a departing path R with branching vertex $b(R)$" means that R is a departing path avoiding some successor of $b(R)$ (to which we do not pay attention). Assume that $p = |V(P_T)|$ is a power of 2 for simplicity. Given $0 \leq i \leq \log p$ and $0 \leq j < 2^i$, we define I^i_j as $I^i_j = P_T[p \cdot j/2^i, p \cdot (j+1)/2^i - 1]$. Intuitively, for each i, $I^i_0, I^i_1, \ldots I^i_{2^i-1}$ form a partition of P_T into 2^i sub-paths of length $p/2^i$ (see Fig. 3). For any sub-path I of P_T, let $\Gamma(I,t)$ be the set of all s-t departing paths R such that $b(R) \in V(I)$ holds. Roughly, the progressive Dijkstra provides a weaker form of the approximate values of $\mathsf{minL}(\Gamma(I^i_j,t))$ for all i and j in $\tilde{O}(m/\epsilon_1)$ time, where "weaker form" means that it computes $\mathsf{minL}(\Gamma(I^i_j,t))$ such that $(1+O(\epsilon_1/\log n)) \cdot \mathsf{minL}(\Gamma(I^i_j,t)) < \min_{0 \leq j' < j} \mathsf{minL}(\Gamma(I^i_{j'},t))$ holds. The intuition of this condition is that the progressive Dijkstra computes $\mathsf{minL}(\Gamma(I^i_j,t))$ only when it is not well-approximated by the length of the best departing path already found. Letting $\mathcal{I}$ be the set of I^i_j for all i and j, for any failed vertex $v_f \in V(P_T)$, the prefix $P_T[0, f-1]$ is covered by a set $C \subseteq \mathcal{I}$ of at most $O(\log p)$ sub-paths in $\mathcal{I}$ (see the grey intervals of Fig. 3). Since the branching vertex of the shortest s-t departing path avoiding v_f obviously lies on $P_T[0, f-1]$, $\mathsf{ddist}_{G-v_f}(s,t) = \min_{I \in C} \mathsf{minL}(\Gamma(I,t))$ holds. By a careful analysis, it is proved that this equality provides a $(1+\epsilon_1)$-approximation of $\mathsf{ddist}_{G-v_f}(s,t)$ by using the outputs of the progressive Dijkstra as approximation of $\mathsf{minL}(\Gamma(I,t))^{\ddagger}$.

[‡] The equality for computing $\mathsf{ddist}_{G-v_f}(s,t)$ actually used in the DP-oracle is further optimized using some additional properties of the progressive Dijkstra, which avoids the explicit construction of C (see

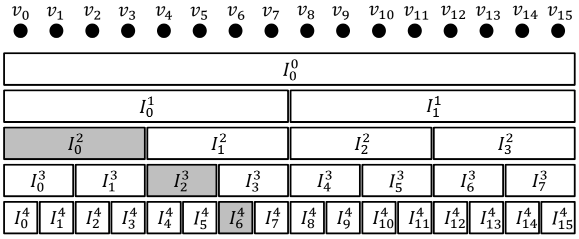

v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13} v_{14} v_{15}
I_0^0
I_0^1 I_1^1
I_0^2 I_1^2 I_2^2 I_3^2
I_0^3 I_1^3 I_2^3 I_3^3 I_4^3 I_5^3 I_6^3 I_7^3
I_0^4 I_1^4 I_2^4 I_3^4 I_4^4 I_5^4 I_6^4 I_7^4 I_8^4 I_9^4 I_{10}^4 I_{11}^4 I_{12}^4 I_{13}^4 I_{14}^4 I_{15}^4

Figure 3 Example of partition of P_T ($p = 16$). The set of grey intervals is an example of C, for vertex v_7.

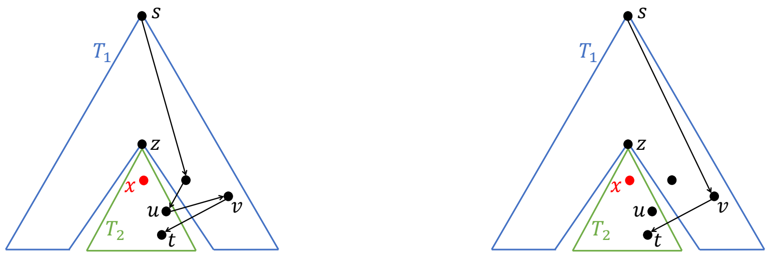
We explain how jumping paths are handled. Let R be the shortest s-t jumping path avoiding $v_f \in V(P_T)$. Since the path from the coalescing vertex $c(b)$ to t in T (of length $\mathsf{dist}_G(c(b), t)$) is not disconnected by v_f, one can assume that the suffix of R from $c(b)$ is the path in T without loss of generality. Then the suffix necessarily contains z, and thus it seems that $\mathsf{jdist}_{G-v_f}(s,t) = \mathsf{dist}_{G-v_f}(s,z) + \mathsf{dist}_T(z,t)$ holds. Unfortunately, it does not always hold, because the shortest s-z replacement path avoiding v_f and the shortest z-t path in T might intersect. However, in such a case, $\mathsf{ddist}_{G-v_f}(s,t) < \mathsf{jdist}_{G-v_f}(s,t)$ necessarily holds and thus the value from the DP-oracle well approximates $\mathsf{dist}_{G-v_f}(s,t)$. Hence it is safe to use the above equality in our case. For computing the approximate values of the right side of the equality for any $v_f \in V(P_T)$ and $t \in V(T_2) \setminus \{z\}$, it suffices to store the values $\mathsf{dist}_{G-v_f}(s,z)$ for all $v_f \in V(P_T)$ approximately. We compute those values using the Bernstein's algorithm [3] which runs in $\tilde{O}(m/\epsilon_1)$ time.

3.2 Construction of G_2

The construction of G_2 for case 2 is relatively simple. Let R be the shortest s-t replacement path from s to $t \in V(T_2)$ avoiding a failed vertex $x \in V(T_2)$. A key observation is that one can assume that R does not contain any backward edge from $V(T_2) \setminus \{z\}$ to $V(T_1)$: if R goes back from $V(T_2)$ to $V(T_1)$ through an edge (u, v), one can replace the prefix of R up to v by the shortest path from s to v in T_1 (See Fig. 4), which never increases the length of R because s-v path in T_1 is the shortest path in G. Notice that the s-v path in T_1 does not contain the failed vertex since $x \in V(T_2) \setminus \{z\}$. This fact implies that R consists of the prefix contained in T_1 and the suffix contained in G_2 which are concatenated by a forward edge from $V(T_1)$ to $V(T_2) \setminus \{z\}$. This observation naturally yields our construction of G_2, where all the vertices in $V(T_1) \setminus \{s\}$ are deleted, and s and each vertex $u \in V(T_2) \setminus \{z\}$ is connected by edge (s, u) of weight $w_{G_2}(s, u) = \mathsf{dist}_{G[\Psi(V(T_1))]}(z, u)$ (See Fig. 5). The weights are computed just by running Dijkstra, which takes $\tilde{O}(m)$ time. Intuitively, the added edge (s, u) corresponds to the prefix of R up to the first vertex in $V(R) \cap V(T_2)$. It should be noted that the edges from s are *safely usable*, that is, the paths corresponding to the added edges do not contain the failed vertex x as stated above. The precise construction of G_2 follow the procedure below:

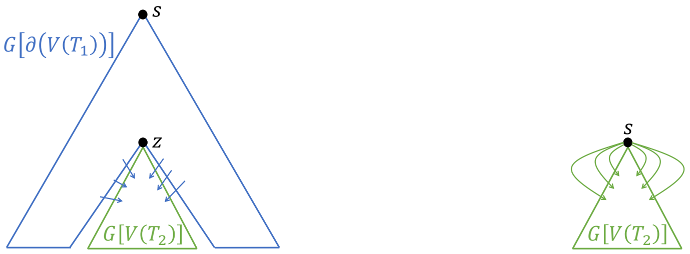
1. $G_2 \leftarrow G[V(T_2) \setminus \{z\}]$
2. Add s to G_2.
3. Calculate the distance $\mathsf{dist}_{G[\Psi(V(T_1))]}(s, u)$ for each vertex $u \in V(T_2) \setminus \{z\}$ by running the standard Dijkstra in $G[\Psi(V(T_1))]$.

Lemma 11 for the details).



(a) The original R avoiding x **(b)** A shorter path avoiding x

Figure 4 An Example of the Case that $E(R)$ contains a backward edge from $V(T_2)$ to $V(T_1)$.



(a) Graph $G[\Psi(V(T_2))]$ (blue subgraph) **(b)** Graph G_2

Figure 5 Graph $G[\Psi(V(T_2))]$ and the construction of G_2.

4. Add the edge (s, u) of weights $\mathsf{dist}_{G[\Psi(V(T_1))]}(s, u)$ to G_2 for each vertex $u \in V(T_2) \setminus \{z\}$ satisfying $\mathsf{dist}_{G[\Psi(V(T_1))]}(s, u) < \infty$.

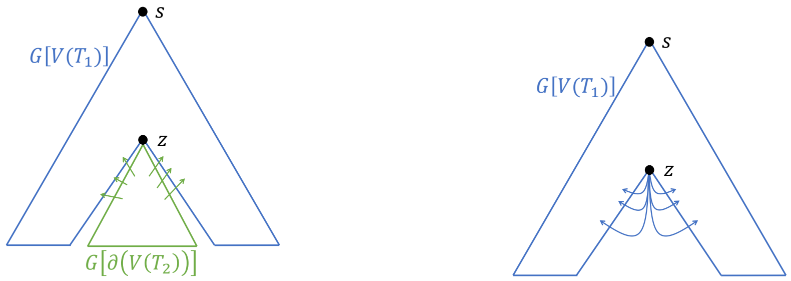
The sub-oracle for G_2 is a $(1 + \epsilon_1)$-VSDO for G_2 and the newly added source s, which is constructed recursively. The correctness of the constructed sub-oracle relies on the following lemma.

▶ **Lemma 6.** *For any failed vertex $x \in V(T_2) \setminus \{z\}$ and destination $t \in V(T_2) \setminus \{z\}$, $\mathsf{dist}_{G_2 - x}(s, t) = \mathsf{dist}_{G-x}(s, t)$ holds.*

3.3 Construction of G_1

We show the construction of G_1 for case 3. If the shortest s-t replacement path avoiding x does not cross $\overline{G}_1 = G[V(T_2) \setminus \{z\}]$, the recursive oracle for $G[V(T_1)]$ well approximates its length. However, paths crossing $\overline{G}_1$ are obviously omitted in such a construction. To handle those omitted paths, we augment $G[V(T_1)]$ with an edge set F, i.e $G_1 = G[V(T_1)] + F$. Let R be the shortest s-t replacement path avoiding x, and assume that R crosses $\overline{G}_1$. The more precise goal is to construct F such that, for any sub-path(s) R' of R whose internal vertices are all in $V(\overline{G}_1)$, there exists a corresponding edge $e \in F$ from the first vertex of R' to the last vertex of R' with a weight approximately equal to $w(R')$. In our construction, the set F is the union of the edge sets F_1 and F_2 provided by two augmentation schemes respectively addressing the two sub-cases of $x \notin V(P_T)$ and $x \in V(P_T)$.

Consider the first scheme which constructs F_1 coping with the sub-case of $x \notin V(P_T)$. We denote by (u, w) the last backward edge from $\overline{G}_1$ to $G[V(T_1)]$ in R (with respect to the order of R). One can assume that the sub-path of R from s to u is the path of T without loss of generality (recall that this sub-path does not contain x by the assumption of $x \in V(T_1)$ and $x \notin V(P_T)$). Then R necessarily contains z. In addition, the length of the sub-path from



(a) Graph $G[\Psi(V(T_2) \setminus \{z\})]$ (green subgraph) **(b)** Graph $G[V(T_1)] + F_1$

Figure 6 Construction of F_1.

z to w is equal to $\mathsf{dist}_{G[\Psi(V(T_2))]}(z, w)$. Hence this case is handled by adding to $G[V(T_1)]$ the edges (z, u) for each $u \in V(T_1)$ of weight $w(z, u) = \mathsf{dist}_{G[\Psi(V(T_2) \setminus \{z\})]}(z, u)$ (See Fig. 6). The weights of added edges are computed just by running Dijkstra with source z in $G[\Psi(V(T_2))]$, which takes $\tilde{O}(m)$ time. Since we call the recursive oracle for G_1 only when the failed vertex x lies in G_1, the paths corresponding to the edges in F_1 are safely usable.

In the sub-case of $x(= v_f) \in V(P_T)$, we only have to consider the situation that R is a jumping-path (i.e., $\mathsf{jdist}_{G-x}(s, t) < \mathsf{ddist}_{G-x}(s, t)$), because the DP-oracle constructed in the first case supports the query for any destination $t \in V(G)$, not only for $t \in V(T_2)$. Then there are the following two possibilities:

- The suffix from $c(R)$ contains a vertex in $V(\overline{G_1})$.
- The prefix of R up to $c(R)$ contains a vertex in $V(\overline{G_1})$.

Note that these two possibilities are not exclusive, and thus both can simultaneously happen. The first possibility is handled by the augmentation with F_1. Let (u, w) be the last backward edge from $\overline{G}_1$ to $G[V(T_1)]$ in R. Due to the case assumption of $\mathsf{jdist}_{G-x}(s, t) < \mathsf{ddist}_{G-x}(s, t)$, one can assume that the sub-path from $c(R)$ to u is the path in T containing z (recall the argument of handling jumping paths in Section 3.1). The edge from z to w corresponding to the sub-path of R from z to w has been already added as an edge in F_1. Hence it suffices to construct F_2 for coping with the second possibility. The baseline idea is to add the edges corresponding to the sub-path of R from $b(R)$ to $c(R)$. Unfortunately, a straightforward application of such an idea for all R (over all possible combinations of v_f and t) results in a too large cardinality of F_2. Instead, we add only a few edges determined by the inside data structure of the DP-oracle which "approximate" the sub-paths from $b(R)$ to $c(R)$ for all possible R. The intuition behind this idea is the observation that the prefix of any jumping path R up to $c(R)$ is a shortest departing path from s to $c(R)$ with branching vertex $b(R)$. That is, the $b(R)$-$c(R)$ sub-path of R is the suffix of $\mathsf{MinP}(\Gamma(I^i_j, c(R)))$ from its branching vertex for some i and j satisfying $b(R) \in I^i_j$. It naturally deduces the construction of F_2 by adding the edges corresponding to the suffixes of the paths computed by the progressive Dijkstra, because those paths well approximate $\mathsf{MinP}(\Gamma(I^i_j, v))$ for all possible triples (i, j, v).

The safe usability of the edges in F_2 is slightly delicate. Assuming the second sub-case of $x \in V(P_T)$, any path Q' corresponding to an edge in F_2 is certainly safe because Q' is the suffix of a departing path Q from $b(Q)$ and thus does not intersect $V(P_T)$. However, in the first sub-case of $x \notin V(P_T)$, Q' might be unavailable, although the corresponding edge in F_2 is still available. Then the recursive oracle for $G_1 = G[V(T_1)] + F$ can return an erroneous shorter length due to the existence of the edges in F_2 if the failed vertex v_f does not lie in P_T. Fortunately, such an error never happens, because if v_f does not lie on P_T, the whole of P_T is available, and using the sub-path from $b(Q)$ to $c(Q)$ of P_T always benefits more than using the added edge $(b(Q), c(Q))$. That is, one can assume that the shortest s-t replacement path does not contain any edge in F_2 without loss of generality if $v_f \notin V(P_T)$ holds.

In summary, the construction of G_1 follows the procedure below.

1. $G_1 \leftarrow G[V(T_1)]$.
2. Calculate the distance $\mathsf{dist}_{G[\Psi(V(T_2))]}(z, u)$ for each vertex $u \in V(T_1)$ by running the standard Dijkstra in $G[\Psi(V(T_2))]$.
3. Add the edge (z, u) of weights $\mathsf{dist}_{G[\Psi(V(T_2))]}(z, u)$ to G_1 for each vertex $u \in V(T_1)$ satisfying $\mathsf{dist}_{G[\Psi(V(T_2))]}(z, u) < \infty$.
4. For each vertex $v_c \in V(P_T)$, $0 \leq i \leq \log p$ and $(\cdot, \ell, v_b) \in \mathsf{upd}(i, v_c)$, add the edge (v_b, v_c) of weights $\ell - \mathsf{dist}_T(s, v_b)$ to G_1.

The sets of the edges added in step 3 and step 4 respectively correspond to F_1 and F_2. If the constructed graph G_1 has parallel edges, only the one with the smallest weight is retained and the others are eliminated. However, for simplicity of the argument, we assume that those parallel edges are left in G_1 without elimination. The correctness of the sub-oracle recursively constructed for G_1 is guaranteed by the following lemma.

▶ **Lemma 7.** *For any failed vertex $x \in V(T_1)$ and destination $t \in V(T_1)$, the following conditions hold:*

- *If $x \notin V(P_T)$, $\mathsf{dist}_{G_1-x}(s, t) = \mathsf{dist}_{G-x}(s, t)$.*
- *If $x \in V(P_T)$ and $\mathsf{jdist}_{G-x}(s, t) < \mathsf{ddist}_{G-x}(s, t)$ holds, $\mathsf{dist}_{G-x}(s, t) \leq \mathsf{dist}_{G_1-x}(s, t) \leq (1 + \epsilon_1) \cdot \mathsf{dist}_{G-x}(s, t)$.*
- *Otherwise, $\mathsf{dist}_{G-x}(s, t) \leq \mathsf{dist}_{G_1-x}(s, t)$.*

Notice that if neither of the first and second conditions are satisfied, the DP-oracle returns the correct approximate distance (recall the discussion in Section 3.3). Then what the sub-oracle for G_1 must avoid is to output an wrongly smaller value. It is ensured by the third condition.

Finally, we remark on the accumulation of approximation factors. Since the weights of edges in F_2 $(1+\epsilon_1)$-approximate the length of the sub-paths from $b(R)$ to $c(R)$, the graph G_1 only guarantees that there exists a s-t path avoiding v_f whose length is $(1+\epsilon_1)$-approximation of $w_G(R)$. Hence even if we construct an $(1 + \epsilon_1)$-VSDO for G_1, its approximation factor as a oracle for G is $(1 + \epsilon_1)^2$. Due to $O(\log n)$ recursion depth, the approximation factor of our oracle finally obtained becomes $(1 + \epsilon_1)^{O(\log n)} = 1 + O(\epsilon_1 \log n)$, which is the reason why we need to set up $\epsilon_1 = \epsilon/(3 \log n)$.

4 Whole Construction and Query Processing

In this section, we summarize the whole construction of $(1 + \epsilon)$-VSDO, and present the details of the query processing. As mentioned in Section 3, we recursively construct the sub-oracles for G_1 and G_2 (for their new sources s). The recursion terminates if the number of vertices in the graph becomes at most 6. At the bottom level, the query is processed by running the standard Dijkstra, which takes $O(1)$ time. Let G^1 be the input graph, and G^{2i} and G^{2i+1} be the graphs G_1 and G_2 for G^i. We denote the shortest path tree of G^i by T_i, its centroid by z_i, and the path from s to z_i in T_i by P_i. We introduce the *recursion tree*, where each vertex is a graph G^i, and G^{2i} and G^{2i+1} are the children of G^i (if they exist). We define $\mathcal{G}_h = \{G^i \mid 2^h \leq i \leq 2^{h+1} - 1\}$, i.e., $\mathcal{G}_h$ is the set of G^i with depth h in the recursion tree. Since the centroid bipartition split T_i into two edge disjoint subtrees of size at most $2n/3$, the recursion depth is bounded by $O(\log n)$, and the total number of recursive calls is $O(n)$. Since one recursive call duplicates the centroid z (into G_1 and G_2), the number of vertices can increase. But the total increase is bounded by $O(n)$, and thus we have $\sum_{G' \in \mathcal{G}_h} |V(G')| = O(n)$. On the increase of edges, the number of edges added to G_1 is bounded by the number of forward edges from $V(T_1)$ to $V(T_2) \setminus \{z\}$. The

Algorithm 1 G^i. **Query**$(\boldsymbol{x}, \boldsymbol{t})$.

```
if $x$ is not an ancestor of $t$ in $T_i$ then
    return $\mathsf{dist}_{T_i}(s,t)$
if $|V(G^i)| \le 6$ then
    return $\mathsf{dist}_{G^i - x}(s,t)$
if $x \in V(P_i)$ and $t \in V(G^{2i+1})$ then
    return $G^i$.pf-query$(x,t)$
if $x \in V(G^{2i})$ and $t \in V(G^{2i})$ then
    return $\min(G^{2i}.\mathsf{dso\text{-}query}(x,t), G^i.\mathsf{dp\text{-}query}(x,t))$
if $x \in V(G^{2i+1})$ and $t \in V(G^{2i+1})$ then
    return $G^{2i+1}$.dso-query$(v_f, t)$
```

number of edges in F_1 is also bounded by the number of backward edges from $V(T_2) \setminus \{z\}$ to $V(T_1)$. Since those forward/backward edges are deleted by the partition, the actual increase is bounded by $|F_2|$, which satisfies $|F_2| = \tilde{O}(n/\epsilon_1)$ (see Lemma 14). Hence we have $\sum_{G' \in \mathcal{G}_h} |E(G')| = \tilde{O}(m + n/\epsilon_1)$. It implies that the total time spent at each recursion level is $\tilde{O}(m/\epsilon_1 + n/\epsilon_1^2)$, and thus the total running time over all recursive calls is $\tilde{O}(m/\epsilon_1 + n/\epsilon_1^2)$. The size of the whole oracle is $\tilde{O}(n/\epsilon_1)$ because at each recursion level the data structure of size $\tilde{O}(n/\epsilon_1)$ is created.

The implementation of dso-query(x,t) is presented in Algorithm 1, which basically follows the recursion approach explained in Section 3. We use notations G^i.dso-query, G^i.pf-query, and G^i.dp-query for clarifying the graph to which the algorithm queries. Since the query processing traverses one downward path in the recursion tree unless x and t is separated by the partition. If $x \in V(G^{2i})$ and $t \in V(G^{2i+1})$ happens, the case 1 of Section 3 (or the trivial case of $x \notin V(P_i)$) applies, and then no further recursion is invoked. The value returned as the answer of G^i.dso-query(x,t) is the minimum of all query responses. It consists of at most $O(\log n)$ calls of pf-query and dp-query. Since each call takes $O(\log n \cdot \log(\epsilon^{-1} \log(nW)))$ time, the query time for dso-query(x,t) is $O(\log^2 n \cdot \log(\epsilon^{-1} \log(nW)))$. In summary, we have the following three lemmas.

▶ **Lemma 8.** *The construction time of the* $(1+\epsilon)$*-VSDO for* G *is* $O(\epsilon^{-1} \log^4 n \cdot \log(nW)(m + n\epsilon^{-1} \cdot \log^3 n \cdot \log(nW)))$ *and the size of the constructed oracle is* $O(\epsilon^{-1} n \log^3 n \cdot \log(nW))$*.*

▶ **Lemma 9.** *For any* $x \in V(G) \setminus \{s\}$ *and* $t \in V(G)$*, the running time of* G*.dso-query*(x,t) *is* $O(\log^2 n \cdot \log(\epsilon^{-1} \log(nW)))$*.*

▶ **Lemma 10.** *For any* $x \in V(G) \setminus \{s\}$ *and* $t \in V(G)$*,* $\mathsf{dist}_{G-x}(s,t) \le$ *dso-query*$(x,t) \le (1+\epsilon) \cdot \mathsf{dist}_{G-x}(s,t)$ *holds.*

The three lemmas above obviously deduce Theorem 1.

5 Concluding Remarks

We presented an algorithm which constructs a $(1+\epsilon)$-VSDO for directed weighted graphs. The constructed oracle attains $\tilde{O}(n \log(nW)/\epsilon)$ size and $\tilde{O}(\log(nW)/\epsilon)$ query processing time. The construction time is $\tilde{O}(m \log(nW)/\epsilon + n \log^2(nW)/\epsilon^2)$. We conclude the paper with a few open questions listed below:

- Can we shave off the extra log factors of our oracle in size and query time, to match them with the best known bounds by [1]?

- Can we extend our result for handling the *dual failure model* admitting two edge/vertex failures? Very recently, the extension of Bernstein's algorithm [3] to the dual failure model has been proposed [13], which attains $\tilde{O}(n^2)$ running time. It is an intriguing open question if there exists an $(1+\epsilon)$-approximate SSRP algorithm for dual failure model running in $\tilde{O}(n^2)$ time or not.
- Is it possible to break the known conditional lower bounds in the case of *additive approximation*?
- Can all-pairs and multi-source cases benefit from the relaxation to $(1+\epsilon)$-approximation?

References

1 Surender Baswana, Keerti Choudhary, Moazzam Hussain, and Liam Roditty. Approximate single-source fault tolerant shortest path. *ACM Trans. Algorithms*, 16(4), July 2020. doi:10.1145/3397532.

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

3 Aaron Bernstein. A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In *Proc. of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 742–755, USA, 2010. doi:10.1137/1.9781611973075.61.

4 Aaron Bernstein and David Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *Proc. of the Forty-First Annual ACM Symposium on Theory of Computing (STOC)*, pages 101–110, 2009. doi:10.1145/1536414.1536431.

5 Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, Simon Krogmann, and Martin Schirneck. Approximate distance sensitivity oracles in subquadratic space. In *Proc. of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1396–1409, 2023. doi:10.1145/3564246.3585251.

6 Davide Bilò, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. Near-Optimal Deterministic Single-Source Distance Sensitivity Oracles. In *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204, pages 18:1–18:17, 2021. doi:10.4230/LIPIcs.ESA.2021.18.

7 Davide Bilo, Luciano Guala, Stefano Leucci, and Guido Proietti. Compact and Fast Sensitivity Oracles for Single-Source Distances. In *24th Annual European Symposium on Algorithms (ESA 2016)*, pages 13:1–13:14, 2016. doi:10.4230/LIPIcs.ESA.2016.13.

8 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-tolerant approximate shortest-path trees. *Algorithmica*, 80(12):3437–3460, 2018. doi:10.1007/s00453-017-0396-z.

9 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. *Algorithmica*, 84(1):37–59, 2022. doi:10.1007/s00453-021-00879-8.

10 Shiri Chechik and Sarel Cohen. Near optimal algorithms for the single source replacement paths problem. In *Proc. of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2090–2109, 2019. doi:10.1137/1.9781611975482.126.

11 Shiri Chechik, Sarel Cohen, Amos Fiat, and Haim Kaplan. $(1+\epsilon)$-approximate f-sensitive distance oracles. In *Proc. of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1479–1496, 2017. doi:10.1137/1.9781611974782.96.

12 Shiri Chechik and Ofer Magen. Near Optimal Algorithm for the Directed Single Source Replacement Paths Problem. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, pages 81:1–81:17, 2020. doi:10.4230/LIPIcs.ICALP.2020.81.

13 Shiri Chechik and Tianyi Zhang. Nearly optimal approximate dual-failure replacement paths. In *Proc. of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2568–2596, 2024. doi:10.1137/1.9781611977912.91.

14 Camil Demetrescu and Mikkel Thorup. Oracles for distances avoiding a link-failure. In *Proc. of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 838–843, 2002. doi:10.1137/S0097539705429847.

15 Dipan Dey and Manoj Gupta. Near optimal algorithm for fault tolerant distance oracle and single source replacement path problem. In *European Symposium on Algorithms (ESA)*, pages 42:1–42:18, 2022. doi:10.4230/LIPIcs.ESA.2022.42.

16 Ran Duan, Yong Gu, and Hanlin Ren. Approximate distance oracles subject to multiple vertex failures. In *Proc. of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2497–2516, 2021. doi:10.1137/1.9781611976465.148.

17 Ran Duan and Hanlin Ren. Maintaining exact distances under multiple edge failures. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1093–1101, 2022. doi:10.1145/3519935.3520002.

18 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 748–757, 2012. doi:10.1109/FOCS.2012.17.

19 Yuzhou Gu, Adam Polak, Virginia Vassilevska Williams, and Yinzhan Xu. Faster Monotone Min-Plus Product, Range Mode, and Single Source Replacement Paths. In *Proc. of 48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, pages 75:1–75:20, 2021. doi:10.4230/LIPIcs.ICALP.2021.75.

20 Manoj Gupta and Aditi Singh. Generic Single Edge Fault Tolerant Exact Distance Oracle. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, pages 72:1–72:15, 2018. doi:10.4230/LIPIcs.ICALP.2018.72.

21 John Hershberger and Subhash Suri. Vickrey prices and shortest paths: What is an edge worth? In *Proc. of 42nd IEEE symposium on foundations of computer science (FOCS)*, pages 252–259, 2001. doi:10.1109/SFCS.2001.959899.

22 John Hershberger, Subhash Suri, and Amit Bhosle. On the difficulty of some shortest path problems. *ACM Trans. Algorithms*, 3(1), 2007. doi:10.1145/1186810.1186815.

23 David R. Karger, Daphne Koller, and Steven J. Phillips. Finding the hidden path: Time bounds for all-pairs shortest paths. *SIAM Journal on Computing*, 22(6):1199–1217, 1993. doi:10.1137/0222071.

24 K. Malik, A.K. Mittal, and S.K. Gupta. The k most vital arcs in the shortest path problem. *Operations Research Letters*, 8(4):223–227, 1989. doi:10.1016/0167-6377(89)90065-5.

25 Enrico Nardelli, Guido Proietti, and Peter Widmayer. A faster computation of the most vital edge of a shortest path. *Information Processing Letters*, 79(2):81–85, 2001. doi:10.1016/S0020-0190(00)00175-7.

26 Enrico Nardelli, Guido Proietti, and Peter Widmayer. Finding the most vital node of a shortest path. *Theoretical Computer Science*, 296(1):167–177, 2003. doi:10.1016/S0304-3975(02)00438-3.

27 Noam Nisan and Amir Ronen. Algorithmic mechanism design (extended abstract). In *Proc. of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, STOC '99, pages 129–140, 1999. doi:10.1145/301250.301287.

28 Merav Parter and David Peleg. Fault-tolerant approximate bfs structures. *ACM Trans. Algorithms*, 14(1), 2018. doi:10.1145/3022730.

29 Merav Parter and David Peleg. Fault tolerant approximate bfs structures with additive stretch. *Algorithmica*, 82(12):3458–3491, 2020. doi:10.1007/s00453-020-00734-2.

30 Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Trans. Algorithms*, 8(4), 2012. doi:10.1145/2344422.2344423.

31 V. Williams, E. Woldeghebriel, and Y. Xu. Algorithms and lower bounds for replacement paths under multiple edge failure. In *Proc. of 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 907–918, 2022. doi:10.1109/FOCS54457.2022.00090.

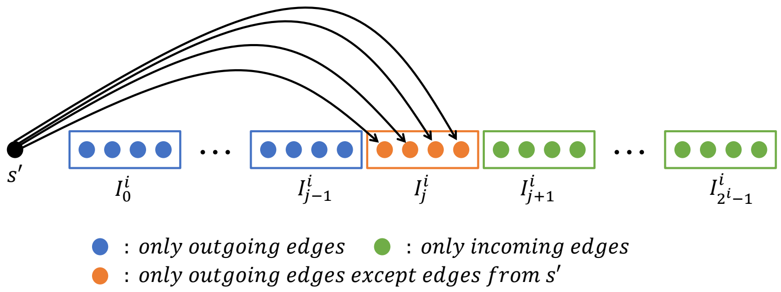


Figure 7 Example of $G_{i,j}$.

32 Virginia Vassilevska Williams. Faster replacement paths. In *Proc. of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1337–1346, 2011. doi:10.1145/3365835.

33 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5), 2018. doi:10.1145/3186893.

A Construction of DP-Oracle

Assume that p is a power of 2 for ease of presentation. To focus on the high-level idea, the detailed proofs of all the lemmas are defferred to the next section. Let $\Pi(v_k,t)$ be the set of all s-t departing paths avoiding v_k, and $\epsilon_2 = \epsilon_1/(2\log n) = O(\epsilon/\log^2 n)$. We define $\Pi(t) = \bigcup_{k\in[0,p-1]} \Pi(v_k,t)$, and introduce the notations $\Gamma(I,t)$, I^i_j, and $\mathcal{I}$ as defined in Section 3. We first present a key technical lemma:

▶ **Lemma 11.** *Let $\Phi(i,j,t)$ be the predicate defined as follows.*

$$\Phi(i,j,t) \Leftrightarrow (j=0) \vee \left((1+\epsilon_2)\cdot \mathsf{minL}(\Gamma(I^i_j,t)) < \min_{0\le j'<j} \mathsf{minL}(\Gamma(I^i_{j'},t)) \right)$$

Assume a function $r : \mathbb{N}\times\mathbb{N}\times V(G) \to \mathbb{R}_+$ satisfying the following two conditions for any $t \in V(G)\setminus\{s\}$ and $0 \le i \le \log p$.

1. $\min_{0\le j'\le j} \mathsf{minL}(\Gamma(I^i_{j'},t)) \le r(i,j,t) \le r(i,j-1,t)$ *for* $j \ge 1$*.*
2. *If $\Phi(i,j,t)$ is true, $r(i,j,t) = \mathsf{minL}(\Gamma(I^i_j,t))$.*

Let $j(i,v_f)$ be the value satisfying $v_f \in I^i_{j(i,v_f)}$. Define $r'(v_f,t)$ as follows.

$$r'(v_f,t) = \min_{0\le i\le \log p} r(i, j(i,v_f)-1, t).$$

Then $\mathsf{minL}(\Pi(v_f,t)) \le r'(v_f,t) \le (1+\epsilon_1)\cdot \mathsf{minL}(\Pi(v_f,t))$ holds for any $v_f \in V(P_T)$ and $t \in V(G)$.

The function r' in Lemma 11 obviously works as a P_T-faulty $(1+\epsilon_1)$-DPO, and the value $r'(v_f,t)$ is easily computed from the function r. Hence the remaining issue is to construct the data structure of returning the value of $r(i,j,t)$ for $0 \le i \le \log p$, $0 \le j < 2^i$, and $t \in V(G)$. The starting point is an algorithm of computing $\mathsf{minL}(\Gamma(I^i_j,v))$ for all $v \in V(G)$. Consider the graph $G_{i,j}$ obtained from G by the following operations (see Fig. 7):

1. Remove all the edges in P_T, all incoming edges of vertices in $V(I^i_0)\cup V(I^i_1)\cup\cdots\cup V(I^i_j)$, and all outgoing edges of vertices in $V(I^i_{j+1})\cup V(I^i_{j+2})\cup\cdots\cup V(I^i_{2^i-1})$.
2. Add a new source vertex s' and edges (s',u) of weights $\mathsf{dist}_{P_T}(s,u)$ for each $u \in I^i_j$.

It is obvious that $\mathsf{dist}_{G_{i,j}}(s', v) = \mathsf{minL}(\Gamma(I^i_{j'}, v))$ holds for any $v \in V(G)$. If we define r as $r(i, j, v) = \min_{0 \leq j' < j} \mathsf{minL}(\Gamma(I^i_{j'}, v))$, the condition of Lemma 11 is satisfied. Hence running Dijkstra in $G_{i,j}$ for all i and j and storing all computation results provides the implementation of the data structure of r. However, the total running time of this approach is expensive, and thus not applicable. For any fixed i, the progressive Dijkstra computes the value $r(i, j, v)$ for all j and v approximately (in the sense of Lemma 11) in $\tilde{O}(m/\epsilon)$ time. Roughly, the progressive Dijkstra iteratively applies a slightly modified version of the standard Dijkstra to $G_{i,j}$ in the increasing order of j. The modified points are summarized as follows:

- When processing $G_{i,0}$, the distance vector d managed by the Dijkstra algorithm initially stores ∞ for all vertices in $V(G)$. When processing $G_{i,j}$ for $j > 0$, the values $d[u]$ for $u \in V(I^i_j)$ are reset to ∞. For all other u, $d[u]$ keeps the results of processing $G_{i,j-1}$.
- All the vertices $u \in V(G) \setminus \{s'\}$ are inactive at the beginning of processing $G_{i,j}$. It becomes active if the algorithm finds a s-u path of length at most $d[u]/(1 + \epsilon_2)$ (i.e., the algorithm finds a path in $G_{i,j}$ whose length is substantially improved from the results for $G_{i,0}, G_{i,1}, \ldots, G_{i,j-1}$). The vertex u is added to the priority queue of the Dijkstra only when it becomes active. After becoming active, the update of $d[u]$ completely follows the standard Dijkstra, i.e., it is updated if the algorithm finds a s'-u path of length less than $d[u]$.
- When $d[u]$ is updated by finding a shorter s'-u departing path Q, the algorithm stores its branching vertex $b(Q)$ into the entry $b[u]$ of vector b.

The whole algorithm runs the above procedure for all $0 \leq i < \log p$. The value $d[v]$ at the end of processing $G_{i,j}$ is used as the value of $r(i, j, v)$. Since it is costly to store those values explicitly, our algorithm stores the value of $r(i, j, v)$ only when $r(i, j-1, v) > r(i, j, v)$ holds. If it holds, v becomes active at processing $G_{i,j}$, and thus $r(i, j-1, v)/(1+\epsilon_2) > r(i, j, v)$ necessary holds. It implies that $r(i, 0, v), r(i, 1, v), \ldots, r(i, 2^i - 1, v)$ store at most $\lceil \log_{1+\epsilon_2}(nW) \rceil$ different values. Those values are stored in the list $\mathsf{upd}(i, v)$. The entry $(j, \ell, b) \in \mathsf{upd}(i, v)$ means that $r(i, j-1, v) > r(i, j, v)$ and $r(i, j, v) = \ell$ hold, and the branching vertex of the corresponding s'-v path is b. Note that the information of the branching vertex is not necessary for constructing the DP-oracle, but required for the construction of G_1. The time cost for accessing the value of $r(i, j, v)$ is $O(\log \log_{1+\epsilon_2}(nW))$, which is attained by a binary search over $\mathsf{upd}(i, v)$. We present the lemma claiming the correctness of this algorithm.

▶ **Lemma 12.** *For any vertex $v \in V(G)$, $0 \leq i \leq \log p$ and $0 \leq j < 2^i$, let us define $r(i, j, v)$ as the value of $d[v]$ after processing $G_{i,j}$. Then, r satisfies the conditions (i), (ii) of Lemma 11.*

The running time of the progressive Dijkstra is bounded by the following lemma.

▶ **Lemma 13.** *The progressive Dijkstra processes $G_{i,j}$ for all $0 \leq j < 2^i$ and $0 \leq i < \log p$ in $O(\epsilon_2^{-1} \cdot \log n \cdot \log(nW) \cdot (m + n \log n))$ time.*

The actual entity of the $(1 + \epsilon_1)$-DPO we construct is upd. For all i and v, $\mathsf{upd}(i, v)$ stores at most $\lceil \log_{1+\epsilon_2}(nW) \rceil$ different values. Hence the total size of the oracle is $O(\epsilon_2^{-1} n \cdot \log n \cdot \log(nW))$. The computation of r' (i.e., processing queries) takes $O(\log p)$ times of accessing upd. Each access takes $O(\log(\epsilon_2^{-1} \log(nW)))$ time, and thus the query time is $O(\log n \cdot \log(\epsilon_2^{-1} \log(nW)))$. By the fact of $\epsilon_2 = O(\epsilon_1/\log n) = O(\epsilon/\log^2 n)$, the following lemma is obtained.

▶ **Lemma 14.** *There exists an algorithm of constructing a P_T-faulty $(1 + \epsilon_1)$-DPO of size $O(\epsilon_1^{-1} n \cdot \log^2 n \cdot \log(nW))$. The construction time is $O(\epsilon_1^{-1} \cdot \log^2 n \cdot \log(nW) \cdot (m + n \log n))$ and the query processing time is $O(\log n \cdot \log(\epsilon_1^{-1} \log(nW)))$.*

A.1 Proofs of Lemma 11 and Lemma 12

We first introduce an auxliliary lemma. For any sub-path I of P_T, a set of sub-paths $C = \{J_0, J_1, \dots, J_{k-1}\} \subseteq \mathcal{I}$ is called a *partition* of I if $V(J_0), V(J_1), \dots, V(J_{k-1})$ exactly cover $V(I)$. The lemma below is easily obtained.

▶ **Lemma 15.** *For any $b \in [0, f-1]$, there exists a partition C of $P_T[0, b-1]$ such that $|C| \leq \lceil \log b \rceil$.*

Now we show the proof of Lemma 11.

Proof. Fix an arbitrary $t \in V(G) \setminus \{s\}$ throughout the proof. By the condition 1, we have

$$r'(v_f, t) \geq \min_{0 \leq i \leq \log p} \min_{0 \leq j' < j(i, v_f)} \mathsf{minL}(\Gamma(I^i_{j'}, t)). \tag{1}$$

Since $\Gamma(I^i_j, t) \subseteq \Pi(v_f, t)$ holds for any $0 \leq i \leq \log p$ and $0 \leq j < j(i, v_f)$, we also have

$$\min_{0 \leq i \leq \log p} \min_{0 \leq j < j(i, v_f)} \mathsf{minL}(\Gamma(I^i_j, t)) \geq \mathsf{minL}(\Pi(v_f, t)). \tag{2}$$

Combining two expressions (1) and (2), we obtain $r'(v_f, t) \geq \mathsf{minL}(\Pi(v_f, t))$. Next, we prove $r'(v_f, t) \leq (1 + \epsilon_1) \cdot \mathsf{minL}(\Pi(v_f, t))$. If $\Phi(i, j, t)$ is true, we say that (i, j) is *strict*. Let $Q = \mathsf{MinP}(\Pi(v_f, t))$ and $v_b = b(Q)$. From Lemma 15, there exists a partition $C = \{J_0, J_1, \dots, J_{|C|-1}\}$ of $P_T[0, b-1]$ with size at most $\lceil \log b \rceil \leq \log p$. For any $0 \leq k < |C|$, let U_k be the prefix $V(J_0) \cup V(J_1) \cup \dots, V(J_k)$ of P_T, and i_k and j_k be the values satisfying $J_k = I^{i_k}_{j_k}$. We also define h as the largest index such that (i_h, j_h) is strict. Note that such h necessary exists because (i_0, j_0) is always strict. By the condition 2, we have,

$$r(i_h, j_h, t) = \mathsf{minL}(\Gamma(I^{i_h}_{j_h}, t)) \leq \mathsf{minL}(\Gamma(U_h, t)), \tag{3}$$

where the right-side inequality comes from the fact that $\Phi(i_h, j_h, t)$ is true. Since $\Phi(i_{h'}, j_{h'}, t)$ is false for any $h' > h$, we also have

$$(1 + \epsilon_2) \cdot \mathsf{minL}(\Gamma(I^{i_{h'}}_{j_{h'}}, t)) \geq \min_{0 \leq j' < j_{h'}} \mathsf{minL}(\Gamma(I^{i_{h'}}_{j'}, t)). \tag{4}$$

Since $V(I^{i_{h'}}_0) \cup V(I^{i_{h'}}_1) \cup \dots \cup V(I^{i_{h'}}_{j_{h'}-1}) = V(U_{h'-1})$ holds, we obtain

$$\min_{0 \leq j' < j_{h'}} \mathsf{minL}(\Gamma(I^{i_{h'}}_{j'}, t)) = \mathsf{minL}(\Gamma(U_{h'-1}, t)). \tag{5}$$

By (4) and (5), we obtain $(1 + \epsilon_2) \cdot \mathsf{minL}(\Gamma(I^{i_{h'}}_{j_{h'}}, t)) \geq \mathsf{minL}(\Gamma(U_{h'-1}, t))$. Further combining this inequality and the fact of $\mathsf{minL}(\Gamma(U_{h'}, t)) = \min\{\mathsf{minL}(\Gamma(U_{h'-1}, t)), \mathsf{minL}(\Gamma(I^{i_{h'}}_{j_{h'}}, t))\}$, we have

$$\mathsf{minL}(\Gamma(U_{h'-1}, t)) \leq (1 + \epsilon_2) \cdot \mathsf{minL}(\Gamma(U_{h'}, t)). \tag{6}$$

In addition, by the definition of Q, $\mathsf{minL}(\Gamma(U_{|C|-1}, t)) = w(Q)$ holds. Utilizing this inequality, (3), and (6), we conclude $r(i_h, j_h, t) \leq (1+\epsilon_2)^{\log p} \cdot w(Q) \leq (1+\epsilon_1) \cdot w(Q)$. Since $r(i_h, j(i_h, v_f) - 1, t) \leq r(i_h, j_h, t)$ holds by the condition 1, we obtain $r'(v_f, t) \leq r(i_h, j_h, t)$. The lemma is proved. ◀

Next, we provide the proof of Lemma 12.

Proof. First, we consider the condition 1. At the end of processing $G_{i,j}$, all the graphs $G_{i,0}, G_{i,1}, \ldots, G_{i,j}$ have been processed. Since $r(i,j,v)$ is not less than the length of the shortest s'-v path in any $G_{i,j'}$ of $0 \leq j' \leq j$, we have

$$r(i,j,v) \geq \min_{0 \leq j' \leq j} \mathsf{dist}_{G_{i,j'}}(s', v) = \min_{0 \leq j' \leq j} \mathsf{minL}(\Gamma(I^i_{j'}, v)).$$

In addition, r is obviously non-increasing with respect to j. Hence the condition 1 is satisfied. Consider the condition 2. For $j = 0$, the standard Dijkstra is executed to process $G_{i,0}$. Hence the condition 2 obviously holds. For $j \geq 1$, $\Phi(i,j,v)$ implies $\mathsf{minL}(\Gamma(I^i_j, v)) = \min_{0 \leq j' \leq j} \mathsf{minL}(\Gamma(I^i_{j'}, v))$. Since $r(i,j,v) \geq \min_{0 \leq j' \leq j} \mathsf{minL}(\Gamma(I^i_{j'}, v))$ holds by the condition 1, $\Phi(i,j,v) \Rightarrow (r(i,j,v) = \mathsf{minL}(\Gamma(I^i_j, v)))$ and $\Phi(i,j,v) \Rightarrow (r(i,j,v) \leq \mathsf{minL}(\Gamma(I^i_j, v)))$ are equivalent. Hence what we show is

$$r(i,j,v) > \mathsf{minL}(\Gamma(I^i_j, v)) \Rightarrow (1 + \epsilon_2) \cdot \mathsf{minL}(\Gamma(I^i_j, v)) \geq \min_{0 \leq j' < j} \mathsf{minL}(\Gamma(I^i_{j'}, v)), \tag{7}$$

which deduces the condition 2. Let $R_{i,j,v} = \langle a_0, a_1, \ldots, a_{k-1} \rangle$ $(s' = a_, v = a_{k-1})$ be the shortest path from s' to v in $G_{i,j}$. By the precondition of the predicate above, $r(i,j,v)$ is not equal to $w(R_{i,j,v})$. It implies that there exists a vertex a_ℓ on $R_{i,j,v}$ which does not become active in processing $G_{i,j}$, because $d[v]$ is correctly updated with the value $w(R_{i,j,v})$ if $a_0, a_1, \ldots, a_{k-1}$ are all active. Without loss of generality, we assume that a_ℓ is such a vertex with the smallest ℓ. Since all the vertices $u \in V(I^i_j)$ are necessarily active due to the reset of $d[u]$, a_ℓ is a vertex not on P_T. At the end of processing $G_{i,j}$, $(1 + \epsilon_2) \cdot w(R_{i,j,a_\ell}) > d[a_\ell] = r(i,j,a_\ell)$ holds. By definition, $w(R_{i,j,a_\ell}) = \mathsf{dist}_{G_{i,j}}(s', a_\ell) = \mathsf{minL}(\Gamma(I^i_j, a_\ell))$ holds, and thus we conclude $r(i,j,a_\ell) < (1 + \epsilon_2) \cdot \mathsf{minL}(\Gamma(I^i_j, a_\ell))$. Let $j' < j$ be the largest value such that a_ℓ becomes active in processing $G_{i,j'}$. Then we have $r(i,j,a_\ell) = r(i,j',a_\ell) = w(R_{i,j',a_\ell})$. We consider a new path Q obtained by concatenating R_{i,j',a_ℓ} and $\langle a_\ell, a_{\ell+1}, \ldots, a_{k-1} \rangle$. The length of Q is bounded as follows:

$$\begin{aligned} w(Q) &= r(i,j',a_\ell) + w(\langle a_\ell, a_{\ell+1}, \ldots, a_{k-1} \rangle) \\ &= r(i,j,a_\ell) + w(\langle a_\ell, a_{\ell+1}, \ldots, a_{k-1} \rangle) \\ &\leq (1 + \epsilon_2) \cdot \mathsf{minL}(\Gamma(I^i_j, a_\ell)) + w(\langle a_\ell, a_{\ell+1}, \ldots, a_{k-1} \rangle) \\ &\leq (1 + \epsilon_2) \cdot \mathsf{minL}(\Gamma(I^i_j, t)). \end{aligned}$$

It is easy to check that Q also exists in $G_{i,j'}$. Hence we have $\min_{0 \leq j' < j} \mathsf{minL}(\Gamma(I^i_{j'}, v)) \leq w(R_{i,j,v}) \leq w(Q)$, i.e., the consequence side of the predicate (7) holds. The lemma is proved. ◀