Improved distance sensitivity oracles via tree partitioning

Ran Duan and Tianyi Zhang

Tsinghua university

Abstract. We introduce an improved structure of *distance sensitivity oracle* (DSO). The task is to pre-process a non-negatively weighted graph so that a data structure can quickly answer replacement path length for every triple of source, terminal and failed vertex. The previous best algorithm [Bernstein and Karger, 2009] constructs in time [1]$\tilde{O}(mn)$ a distance sensitivity oracle of size $O(n^2 \log n)$ that processes queries in $O(1)$ time. As an improvement, our oracle takes up $O(n^2)$ space, while preserving $O(1)$ query efficiency and $\tilde{O}(mn)$ preprocessing time. One should notice that space complexity and query time of our novel data structure are asymptotically optimal.

1 Introduction

The objective of *distance sensitivity oracles* (DSO) is to pre-process the graph so that pairwise shortest distances can be answered by a static data structure in constant time even when one vertex or edge crashes. More precisely, every query for the DSO is composed of three fields: a source vertex, a terminal vertex, and a vertex or an edge that is presumed failed, and then the DSO is supposed to compute the length of shortest source-terminal replacement path that circumvents the failed vertex or edge. In this paper, we are only concerned with vertex failures, as all vertex-failure DSOs can be easily extended to handle edge-failure queries without any asymptotic loss in space / time efficiency [10].

Motivations for distance sensitivity oracles mainly come from practical scenarios like network routing where some nodes occasionally undergo crash failures. As recomputing all-pair shortest paths from scratch every time a node or link crashes is expensive, a static data structure like DSOs that plans for emergency is highly desirable. Vickery pricing [19] is an example that motivates the DSOs from a theoretical perspective, where one wishes to measure, for every pair of source and target as well as a failed edge, by how much the shortest distance would rise if this designated edge were to shut down.

1.1 Related work

The naive approach is that we pre-compute and store the length of all $O(n^3)$ possible replacement path lengths, which incurs intolerable space complexity.

[1] $\tilde{O}(\cdot)$ suppresses poly-logarithmic factors.

F. Ellen et al. (Eds.): WADS 2017, LNCS 10389, pp. 349–360, 2017.
DOI: 10.1007/978-3-319-62127-2_30

Authors of [12] proposes the first DSO that occupies only near-quadratic space. More specifically, The DSO in [12] has space complexity $O(n^2 \log n)$ and $O(1)$ query time. Space complexity of constant query time DSO has not been improved ever since.

DSO in [12] demands a somewhat high preprocessing time complexity of $O(mn^2)$, which was improved to $\tilde{O}(mn^{\frac{3}{2}})$ in the journal version [10] while the space complexity was blown up to $O(n^{2.5})$. Cubic time preprocessing algorithm was first obtained by [6] and shortly improved from $\tilde{O}(n^2\sqrt{m})$ to $\tilde{O}(mn)$ in [7], while maintaining $O(n^2 \log n)$ space and $O(1)$ query time. Note that $O(n^2 \log n)$ and $\tilde{O}(mn)$ are basically optimal up to poly-logarithmic factors, as discussed in [7]. Therefore, surpassing [7]'s construction time has been deemed hard from then.

Since the publication of [7], the community's interest has diverged to seeking truly sub-cubic preprocessing time algorithms. F. Grandoni and V. Williams [15] obtained truly sub-cubic preprocessing time bound $O(Mn^{2.88})$, if one should tolerate a sub-linear query time of $O(n^{0.7})$; here all edge weights are assumed to be integers within interval $[-M, M]$.

There are several generalizations of the distance sensitivity problem. In [13], the authors considered the scenario where two instead of one vertices could fail. The paper presented a distance oracle with $O(n^2 \log^3 n)$ space complexity and $O(\log n)$ query time. As it turned out, things got far more complicated than single vertex-failures, and sadly no non-trivial polynomial preprocessing algorithms were known.

There are papers (e.g., [22, 9, 11, 3, 2]) mainly concerned with dynamically maintaining all pairs shortest paths (APSP). Such data structures can solve distance problems if subsequent failures are cumulative, but update time would be as large as $O(n^{\frac{8}{3}})$.

If one should sacrifice preciseness for space efficiency, one may consider approximate distance oracles for vertex failures. Authors of [20] considered approximating the replacement path lengths where a single vertex could crash. In [8], the authors focused on data structures that approximately answer minimal length of paths that do not pass through a designated set of failed edges. For fully dynamic approximate APSP, one can refer to [16]; especially, for planar graphs, [1] may provide useful results.

There are remotely related problems such as (partially) dynamic single-source shortest paths and reachability. Papers [18] and [17] discussed these topics in depth. Some other loosely related work concerns the construction of spanners and distance preservers resilient to one edge / node failure.

1.2 Our result

In this paper we present a DSO construction that improves upon [7].

Theorem 1. *For any directed non-negatively weighted graph $G = (V, E)$ with weight function $\omega : E \to R^+ \cup \{0\}$, with m, n referring to total number of edges*

and vertices, a DSO with $O(n^2)$ space complexity and $O(1)$ query time exists. Also, such DSO can be preprocessed in time $\tilde{O}(mn)$.

In Bernstein & Karger's work [7], the space / query time was $O(n^2 \log n)$ and $O(1)$. So compared with [7]'s result, our construction shaves off the last $\log n$ factor in the space complexity, leading to a quadratic space consumption, while preserving constant time query efficiency. Plus, our DSO can also be constructed in $\tilde{O}(mn)$ time as in [7], which is nearly optimal. We emphasize that the space complexity of our DSO is asymptotically **optimal** even for sparse graphs, assuming hardness of set intersection [21].

The observation comes from Demetrescu and Thorup's original work [12]. Its construction basically applies the idea of sparse table, where each pair of source and target are associated with $O(\log n)$ sparse table entries, thus resulting in a total storage of $O(n^2 \log n)$. Our preliminary idea is that we do not store sparse table entries for every source-terminal pair. More specifically, for each single source shortest path (SSSP) tree, only a proportion of all tree vertices will be associated with sparse table entries, hence making our data structure even sparser. The set of all designated vertices should be carefully chosen with respect to the topological structure of the SSSP tree.

With the sparser data structure, we can answer queries (s, t, f) when f keeps distance from both of s and t. So the bottleneck lies in degenerated cases where f is very close to one of the endpoints s or t. For degenerated cases, we can use much smaller sparse tables to cover short paths, resulting in a DSO that only occupies $O(n^2 \log \log n)$ space. To obtain an optimum of $O(n^2)$ space complexity, we would need to apply a tabulation technique (otherwise known as the "Four Russians" [4]).

Our preprocessing algorithm heavily relies on the notion of *admissible functions* from [6]. The idea is that we substitute "bottleneck" vertices for intervals in the original construction, without harming the correctness of query algorithm. This would largely facilitate the preprocessing algorithm of DSOs.

2 Preliminaries

Suppose we are given a directed graph $G = (V, E)$ with non-negative edge weights $\omega : E \to R^+ \cup \{0\}$. In this section, we summarize the notations or assumptions that are used throughout this paper. More or less, we inherit the conventions from [13].

- Our data structures & algorithms are implemented on $\Omega(\log n)$-word RAM machines. We will leverage its strength in computing the most significant set bit [14]. Although in previous works on DSO ([9, 10, 6, 7]) $\Omega(\log n)$-word RAM model was not explicitly assumed, this assumption is not dispensable in these papers since their algorithms required memory indexing and computing logarithms in constant time.
- We call a data structure $\langle f(n), g(n) \rangle$, if its space complexity is at most $f(n)$ and its query time is at most $g(n)$.

- For each pair of $s, t \in V$, the weighted shortest path from s to t is unique. This assumption is without loss of generality since we can add small perturbations to break ties (e.g., [10, 7]).
- For each pair of $s, t \in V$, let st denote the weighted shortest path from s to t.
- Let p be a simple path. Denote by $\|p\|$ and $|p|$ the weighted and un-weighted length of path p.
- For each $s \in V$, let T_s be the single-source shortest path tree rooted at s; let $\widehat{T}_s$ be the single-source shortest path tree rooted at s in the reverse graph $\widehat{G}$ where every directed edge in G is reversed.
- For each query (s, t, f), we only consider the case when f lies on the path st, because verification can be done by checking whether $\|st\| = \|sf\| + \|ft\|$.
- For each vertex set A, let $st \diamond A$ denote the weighted shortest path from s to t that avoids the entire set A. For instance, $st \diamond \{f\}$ (abbreviated as $st \diamond f$) denotes the replacement path, and $st \diamond [u, v]$ refers to the weighted shortest path that skips over an entire interval $[u, v] \subseteq st$. Here for $[u, v]$ to be a properly defined interval on st, it is required that both of u and v are on st, and either $u = v$, or u lying between s and v.
- Let $s \oplus i$ and $s \ominus i$ be the i^{th} vertex after and before s on some path that can be learnt from context.

By uniqueness of shortest paths, it is easy to verify that any path of the form $st \diamond f$ ($st \diamond [u, v]$) must diverge from and converge with st for **at most** once [10], with divergence on path sf (su) and convergence on ft (vt). We denote Δ and ∇ to be the vertices at which divergence and convergence take place, respectively, when the path can be learnt from context.

3 Admissible functions and the triple path lemma

Our query algorithms will be frequently using the *triple path lemma* from [6]. This lemma is based on the notion of *admissible functions*.

Definition 1 ([6])**.** A function $F_{s,t}^{[u,v]}$ is admissible if $\forall f \in [u, v]$, $[u, v]$ an interval on st, we have $\|st \diamond [u, v]\| \geq F_{s,t}^{[u,v]} \geq \|st \diamond f\|$.

Definition 2 ([6])**.** Two important admissible functions are $\max_{f \in [u,v]}\{\|st \diamond f\|\}$ and $\|st \diamond [u, v]\|$. We call them **bottleneck** and **interval** admissible functions, respectively.

Lemma 2 (The triple path lemma [6])**.** *Let $[u, v]$ be an interval on st, and $f \in [u, v]$ be a vertex. For any admissible function $F_{s,t}^{[u,v]}$, $\|st \diamond f\| = \min\{\|su\| + \|ut \diamond f\|, \|sv \diamond f\| + \|vt\|, F_{s,t}^{[u,v]}\}$.*

Proof. On the one hand, $\|st \diamond f\|$ is always smaller or equal to $\min\{\|su\| + \|ut \diamond f\|, \|sv \diamond f\| + \|vt\|, F_{s,t}^{[u,v]}\}$. This is because, by definition 1 $F_{s,t}^{[u,v]} \geq \|st \diamond f\|$; also it is easy to see that both of $\|su\| + \|ut \diamond f\|$ and $\|sv \diamond f\| + \|vt\|$ are $\geq \|st \diamond f\|$.

On the other hand, we argue $\|st \diamond f\| \geq \min\{\|su\| + \|ut \diamond f\|, \|sv \diamond f\| + \|vt\|, F_{s,t}^{[u,v]}\}$. If $st \diamond f$ passes through either u or v, then $\|st \diamond f\|$ would be equal to either $\|su\| + \|ut \diamond f\|$ or $\|sv \diamond f\| + \|vt\|$, which is $\geq \min\{\|su\| + \|ut \diamond f\|, \|sv \diamond f\| + \|vt\|, F_{s,t}^{[u,v]}\}$. Otherwise $st \diamond f$ skips over the entire interval $[u, v]$ and thus $\|st \diamond f\| = \|st \diamond [u, v]\| \geq F_{s,t}^{[u,v]} \geq \min\{\|su\| + \|ut \diamond f\|, \|sv \diamond f\| + \|vt\|, F_{s,t}^{[u,v]}\}$.

4 The tree partition lemma

Our novel DSO begins with the following lemma. Paper [15] has a similar lemma, but it is actually different from ours.

Lemma 3 (Tree partition)**.** *Given a rooted tree $\mathcal{T}$, and any integer $2 \leq k \leq n = |V(\mathcal{T})|$, there exists a subset of vertices $M \subseteq V(\mathcal{T})$, $|M| \leq 3k-5$, such that after removing all vertices in M, the tree $\mathcal{T}$ is partitioned into sub-trees of size $\leq n/k$. We call every $u \in M$ an M-marked vertex, and M a marked set. Plus, such M can be computed in $O(n \log k)$ time.*

A detailed proof can be found in the full version of this paper.

The high-level idea of our data structure is that we reduce the computation of an arbitrary $\|st \diamond f\|$ to a "shorter" $\|uv \diamond f\|$; here we say "short" in the sense that either $|uf|$ or $|fv|$ is small. The tree partition lemma helps us with the reduction. Basically, we apply the lemma **twice** with different parameters so that either uf or fv becomes "short" enough, and then we can directly retrieve the length of replacement path from storage.

4.1 Shortness of replacement paths

Our new data structure will heavily rely on the notion of "shortness" of replacement paths, which we describe below. On a high-level, for any replacement path $st \diamond f$, shortness measures how close f is to either s or t on the weighted shortest path st. For those replacement paths where f lies near to the middle of st, we think of them as long ones, and for those where f is very close to one of the endpoints, we view them as short paths. The rough idea is that, when we compute the length of a general replacement path, we reduce it to a constant number of shorter paths and conquer them separately. Now we propose a formal definition of shortness.

Definition 3. Given a vertex subset M, whose removal breaks T (either T_s or $\hat{T}_s$) into subtrees of size $< L$, L being a fixed parameter. Then for any t, as well as a failed vertex f on path st, we say $st \diamond f$ is *L-short* with respect to M, if t and f lie in the same subtree of T induced by removal of M. We often do not explicitly refer to M when it can be learnt from context.

5 Reducing to $\log^2 n$-short paths

We devise an $O(n^2)$-space data structure that computes any non-$\log^2 n$-short path in constant time.

5.1 Data structure

Our DSO first pre-computes all values of $\|st\|$ and $|st|$, which accounts for $O(n^2)$ space. Then, for each $s \in V$, apply lemma **3** in T_s ($\widehat{T}_s$) to obtain a marked set M_s ($\widehat{M}_s$) with parameter $\lceil n/(L-1) \rceil$, where $2 \leq L \leq n$ is an integer to be set later. So M_s ($\widehat{M}_s$) is of size $O(n/L)$, and the size of each sub-tree is $< L$. Consider the following structures.

For any pair of s, t such that $t \in M_s$, suppose we are met with M_s-marked vertices $u_1 \to u_2 \to \cdots\cdots \to u_k = t$ along the path st in T_s. Our data structure consists of several parts. Note that we also build the symmetric structures of (i) to (iv) for every pair of s, t where $t \in \widehat{M}_s$.

(i) For each $k - 2^i \in [1, k-1]$, the value of $\|st \diamond u_{k-2^i}\|$.
(ii) For each $k - 2^i \in [1, k-2]$, the value of $\|st \diamond [u_{k-2^i}, u_{k-2^i+1}]\|$.
(iii) Let $v_l \to \cdots\cdots \to v_1$ be the sequence of all $\widehat{M}_t$-marked vertices along the path st. Then for each properly defined interval $[v_{l-2^i}, u_{k-2^j}]$ on the path st, store the value of $\|st \diamond [v_{l-2^i}, u_{k-2^j}]\|$.
(iv) For each f such that $|ft| \leq 2L$ or $|sf| \leq 2L$, the value of $\|st \diamond f\|$.
From now on we drop the assumption that t is M_s-marked.
(v) For every pair (s, t) of different vertices, let x be t's nearest M_s-marked T_s-ancestor, and y be s's nearest $\widehat{M}_t$-marked $\widehat{T}_t$-ancestor. If intervals $(s, x]$ and $[y, t)$ intersect, then we pre-compute and store $\|st \diamond [y, x]\|$. Also, store addresses of x, y, if such ancestors exist.
(vi) Build a tree upon all $M_s \cup \{s\}$'s vertices as follows. In this tree, u is v's parent if and only if in T_s u is v's nearest ancestor that belongs to $M_s \cup \{s\}$. Then pre-compute and store the level-ancestor [5] data structure of this tree. Note again that we also build similar structures for $\widehat{M}_s \cup \{s\}$ in the reverse graph.

Note 1. The un-weighted distances between two adjacent marked vertices in T_s ($\widehat{T}_s$) are $\leq L$.

Conduct a simple space complexity analysis for each part of the data structure.

(i) takes up space $O(\frac{n^2 \log n}{L})$ since we have $O(\log n)$ choices for the index i, and every $|M_s| = O(n/L)$.
(ii) uses $O(\frac{n^2 \log n}{L})$ for a similar reason as in the previous part.
(iii) demands $O(\frac{n^2 \log^2 n}{L})$ space since we have $O(\log^2 n)$ choices for the pair of (i, j).
(iv) entails an $O(n^2)$ space consumption since each M_s-marked t is associated with $O(L)$ entries, and there are $O(n/L)$ M_s-marked vertices t.
(v) induces $O(n^2)$ space complexity.
(vi) takes $O(n^2/L)$ total space, each tree of size $O(n/L)$.

Therefore, the overall space complexity from (i) through (vi) is equal to $O(\frac{n^2 \log^2 n}{L} + n^2)$. Taking $L = \log^2 n$, it becomes $O(n^2)$.

5.2 Query algorithm

We prove the following reduction lemma in this sub-section.

Lemma 4. *The data structure specified in section **5.1** can compute $\|st \diamond f\|$ in $O(1)$ time if $st \diamond f$ is not L-short with respect to M_s or $\widehat{M}_t$.*

Proof. A constant time verification for L-shortness is easy: we check if f lies below x, which is the nearest M_s-marked T_s-ancestor of t, and similarly if f lies below y, which is the nearest $\widehat{M}_t$-marked $\widehat{T}_t$-ancestor of s.

Firstly we argue that it is without loss of generality to assume that t is M_s-marked. The reduction proceeds as follows.

Let x and y be vertices defined as in (v) from section **5.1**. Since $st \diamond f$ is not L-short, $f \in (s, x] \cap [y, t)$, and thus $[y, x]$ is a properly defined interval on path st. By the *triple path lemma*, one has:

$$\|st \diamond f\| = \min\{\|sx \diamond f\| + \|xt\|, \|sy\| + \|yt \diamond f\|, \|st \diamond [y, x]\|\}$$

Here we use the interval admissible function $\|st \diamond [y, x]\|$. Noticing that the third term $\|st \diamond [y, x]\|$ is already covered in (v) from section **5.1**, we are left with $\|sx \diamond f\|$ and $\|yt \diamond f\|$. By definition, x is M_s-marked and y is $\widehat{M}_t$-marked, and thus we complete our reduction.

Let $u_1 \to u_2 \to \cdots\cdots \to u_p = t$ be the sequence of all M_s-marked vertices along st. We can assume that $p > 1$; otherwise $\|st \diamond f\|$ has already been computed in structure (iv).

It is not hard to find the interval $[u_a, u_{a+1})$ that contains f. On the one hand, u_a is easily retrieved: if f not M_s-marked, then u_a is its nearest marked ancestor stored in (v); otherwise, $u_a = f$. On the other hand, u_{a+1} can be found by querying the level-ancestor data structure (vi) at node t in tree rooted at s.

Let $v_q \to v_{q-1} \to \cdots\cdots \to v_1$ be the sequence of all $\widehat{M}_t$-marked vertices on st. It is also safe to assume $q > 1$; otherwise, we have $|st| \leq 2L$ and then using (iv) we can directly compute $\|st \diamond f\|$. Similar to the previous paragraph, locate the interval $(v_{b+1}, v_b]$ that includes f. We only need to consider the case when $a + 1 < p$ and $b + 1 < q$, since otherwise $\|st \diamond f\|$ can be directly retrieved from structure (iv).

Find maximum indices $i, j \geq 0$ such that $q - 2^i \geq b + 1$, $p - 2^j \geq a + 1$. Applying the *triple path lemma* with respect to $[v_{q-2^i}, u_{p-2^j}]$ in terms of interval admissible function, $\|st \diamond f\|$ must be the minimum among the following three distances.

(1) $\|st \diamond [v_{q-2^i}, u_{p-2^j}]\|$.
This value is directly retrievable from (iii) in section **5.1**.

(2) $\|su_{p-2^j} \diamond f\| + \|u_{p-2^j}t\|$.
Note that since we are interested in the minimum among (1)(2)(3) which gives us $\|st \diamond f\|$, we can substitute any value for (2) that lies in range $[\|st \diamond f\|, \|su_{p-2^j} \diamond f\| + \|u_{p-2^j}t\|]$.

By definition of j, u_{a+2^j} lies between u_{p-2^j} and t, and hence we know that the concatenation of paths $su_{p-2^j} \diamond f$ and $u_{p-2^j}t$ passes through u_{a+2^j}. Thus, it must be

$$\|su_{p-2^j} \diamond f\| + \|u_{p-2^j}t\| \geq \|su_{a+2^j} \diamond f\| + \|u_{a+2^j}t\| \geq \|st \diamond f\|$$

So instead of computing the original (2), we are actually calculating $\|su_{a+2^j} \diamond f\| + \|u_{a+2^j}t\|$.

We focus on the case when $f \neq u_a$; the case where $f = u_a$ is easy in that we can directly query $\|su_{a+2^j} \diamond u_a\|$ using (i).

Applying the *triple path lemma* for a third time to $su_{a+2^j} \diamond f$ and interval $[u_a, u_{a+1}]$, we further divide it into three cases.

(a) $\|su_{a+2^j} \diamond [u_a, u_{a+1}]\| + \|u_{a+2^j}t\|$.
This can be computed by a single table lookup in (ii).

(b) $\|su_{a+1} \diamond f\| + \|u_{a+1}t\|$.
Since u_{a+1} is M_s-marked, $\|su_{a+1} \diamond f\|$ is stored in structure (iv), and thereby $\|su_{a+1} \diamond f\| + \|u_{a+1}t\|$ is computed effortlessly.

(c) $\|su_a\| + \|u_at \diamond f\|$.
If u_a itself is $\widehat{M}_t$ marked, then (iv) directly help us out since $\|u_at \diamond f\|$ is already pre-computed as $|u_af| \leq L$.
Otherwise, suppose v is u_a's nearest $\widehat{M}_t$-marked ancestor in T_s (if any). To locate such v, we can try to find the interval $(v_{c+1}, v_c]$ that contains u_a, in a similar fashion of finding intervals $[u_a, u_{a+1})$ and $(v_{b+1}, v_b]$; after that we assign $v \leftarrow v_{c+1}$.
If such v does not exist, then s and u_a lie in the same sub-tree of $\widehat{T}_t$ after removing $\widehat{M}_t$. Noticing that $|sf| < |su_{a+1}| = |su_a| + |u_au_{a+1}| \leq 2L$, (iv) can finish up $\|st \diamond f\|$ by a single table look-up. If v exists, then by $\|su_a\| + \|u_at \diamond f\| \geq \|sv\| + \|vt \diamond f\| \geq \|st \diamond f\|$, it suffices to compute $\|vt \diamond f\|$, which also has already been pre-computed in (iv) due to $|vf| = |vu_a| + |u_af| \leq 2L$.

(3) $\|sv_{q-2^i}\| + \|v_{q-2^i}t \diamond f\|$.
The only difficult part is $\|v_{q-2^i}t \diamond f\|$. Similar arguments as in the previous case (2) would still work.

6 An $\langle O(n^2 \log\log n), O(1)\rangle$ construction

In this section, we present an ordinary way of handling L-short paths, resulting in an $\langle O(n^2 \log\log n), O(1)\rangle$ DSO. On a high level, we directly apply the sparse table construction as in [10]. But since the sparse table only needs to cover L-short paths, the space requirement shrinks to $O(\log L) = O(\log\log n)$ for every pair of $s, t \in V$. Hence the total space complexity would be $O(n^2 \log\log n)$.

6.1 Data structure

For any pair of $s, t \in V$, besides $\|st\|, |st|$, build the following structures.

(i) For every $2^i \leq \min\{4L, |st| - 1\}$, store $\|st \diamond (s \oplus 2^i)\|$ and $\|st \diamond (t \ominus 2^i)\|$.
(ii) For every $2^{i+1} \leq \min\{4L, |st| - 1\}$, store $\|st \diamond [s \oplus 2^i, s \oplus 2^{i+1}]\|$ and $\|st \diamond [t \ominus 2^{i+1}, t \ominus 2^i]\|$.
(iii) Level ancestor data structures of T_s and $\widehat{T}_s$.

Since $L = \log^2 n$, the total space of this structure is equal to $O(n^2 \log L) = O(n^2 \log\log n)$. Note that this structure is basically identical to [10], except for the additional bound $4L$ on the power-of-two's 2^i. Therefore, when $|st| \leq 4L$, $\|st \diamond f\|$ can be retrieved in $O(1)$ time according to the correctness guaranteed by [10].

6.2 Query algorithm

We prove the following lemma, showing how our data structure covers all L-short paths.

Lemma 5. *For any $st \diamond f$ such that $|sf| \leq L$ or $|ft| \leq L$, $\|st \diamond f\|$ can be computed in $O(1)$ time by the data structure presented in section **6.1**.*

For details of the proof, please refer to the full version of this paper.

From definition **3**, we can see every L-short path $st \diamond f$ satisfies $|sf| \leq L$ or $|ft| \leq L$. So by the above lemma, data structure introduced in section **6.1** answers every L-short path query. Together with the structures in section **5.1**, it makes an $\langle O(n^2 \log\log n), O(1)\rangle$ DSO.

7 Two-level partition

In this section, we obtain an $\langle O(n^2), O(1)\rangle$ construction of DSO. The high-level idea is that we further partition every sub-tree into even smaller ones, and then we apply a tabulation ("Four Russians") technique to store all answers. More specifically, we apply the tree-partitioning lemma for the second time and break each SSSP tree into sub-trees of size $\leq \log\log^2 n$. Then we devise data structures to reduce $\log^2 n$-short paths to $\log\log^2 n$-short paths. Finally, the tabulation technique kicks in when it comes to $\log\log^2 n$-short paths.

7.1 Data structure

Let $L' \leq L$ be a parameter to be set later. For each SSSP tree T_s ($\widehat{T}_s$), compute its tree partition with parameter $\lceil n/(L'-1)\rceil$ by Lemma 3, and let M'_s ($\widehat{M}'_s$) be the corresponding marked set. For any t, let r be the root of the sub-tree, induced by removal of M_s, that contains vertex t. We build the following data structures. (Note that similar structures are also built for the reverse graph where t is the source and s is the terminal.)

(i) If t is not M'_s-marked.
Let u be t's nearest M'_s-marked ancestor below r (if such u exists), and store the value of $\|st \diamond [r, u]\|$.

(ii) If t is M'_s-marked.
Let $u_1 \to u_2 \to \cdots\cdots \to u_k = t$ be the sequence of all M'_s-marked ancestor along the directed path rt. Note that $k < L = O(\log^2 n)$. Then for each $k - 2^i \in [1, k-1]$, store the value of $\|st \diamond [r, u_{k-2^i}]\|$. After that, for each $f \in [u_{k-1}, u_k)$ (define $u_0 = r$), store the value of $\|st \diamond f\|$.

(iii) Build upon the marked set M'_s all structures from (i) to (vi) in section **5.1**. The only difference is that we impose an additional constraint that $|st| \leq L$ on structures (i) through (iii). It is not hard to verify that the space complexity of this part becomes $O(\frac{n^2 \log^2 L}{L'} + n^2) = O(\frac{n^2 \log\log^2 n}{L'} + n^2)$. So if $st \diamond f$ is non-L'-short, with $|st| \leq L$, then applying lemma **5.1**, $\|st \diamond f\|$ can be answered in constant time.

Note that the space complexity of (i) and (ii) in section **7.1** is equal to $O(\frac{n^2 \log\log n}{L'} + n^2)$. Together with (iii), the overall space complexity of the data structure is $O(n^2 + \frac{n^2 \log\log n}{L'} + \frac{n^2 \log\log^2 n}{L'})$. Taking $L' = \log\log^2 n$, the space becomes $O(n^2)$.

7.2 Reduction algorithm

We claim the following lemma; due to page limits, its full proof is presented in the full version.

Lemma 6. *Given an L-short replacement path $st \diamond f$, the data structure in sections **7.1** and **5.1** can reduce $\|st \diamond f\|$ to a constant number of $\|uv \diamond f\|$'s, where $uv \diamond f$'s are L'-short with respect to M'_u or $\widehat{M}'_v$.*

7.3 Tabulation

In this sub-section, we handle all L'-short paths. Recall that the notation Δ, ∇ refers to divergence and convergence of replacement path $st \diamond f$. when s, t, f can be learnt from context.

Let $st \diamond f$ be an L'-short path; without loss of generality assume that t, f lie in the same sub-tree, the corresponding marked set being M'_s. One observation is that we only need to focus on cases where $|s\Delta| \leq L$: if the divergence comes after $s \oplus L$, then it admits the decomposition $\|st \diamond f\| = \|su\| + \|ut \diamond f\|$, u being s's nearest $\widehat{M}_t$-marked ancestor in tree $\widehat{T}_t$. Since $|ft| \leq L' < L < 2L$, $\|ut \diamond f\|$ can be found in (iv) from section **5.1**.

For each sub-tree T partitioned by marked set M'_s, we in-order sort all its vertices. The aggregate divergence / convergence information within this sub-tree can be summarized as an $L' \times L'$ matrix, each element being a pair $(|s\Delta|, |\nabla t|)$ corresponding to a replacement path $st \diamond f$, $\forall t, f \in V(T)$. Since we only consider the case when $|s\Delta| \leq L$, the total number of choices for this matrix is no more than $(L \cdot L')^{(L')^2} < L^{2(L')^2} = 2^{4\log\log^5 n} = o(n)$. Recall we are running on $\Omega(\log n)$-word RAM machines, so this matrix admits random accesses.

Construct an indexable table of all possible configurations of such matrices. The space of this table is $\leq o(n \cdot (L')^2) = o(n \log\log^2 n) = o(n^{1.1})$. Then associate

each sub-tree with an index of its corresponding matrix in the table, which demands a storage of $O(n/L')$ indices, totalling $o(n)$ space for every s. Thus the overall space complexity associated with tabulation is $o(n^2)$.

Now the L'-short $\|st \diamond f\|$ can be computed effortlessly. After indexing the corresponding matrix in the table, we can extract $(|s\Delta|, |\nabla t|)$ directly from this matrix, and then recover Δ, ∇ from level-ancestor data structures. Finally, decompose the replacement path as $\|st \diamond f\| = \|s\Delta\| + \|\Delta\nabla \diamond (\Delta, \nabla)\| + \|\nabla t\|$. Noticing that $\Delta\nabla \diamond f = \Delta\nabla \diamond (\Delta, \nabla)$, thereby the value of $\|\Delta\nabla \diamond f\|$ is equal to any admissible function value $F_{\Delta,\nabla}^{[\Delta\oplus 1, \nabla\ominus 1]}$. Hence, storing a $\|uv \diamond [u\oplus 1, v\ominus 1]\|$ for every pair of u, v will suffice for querying $\|st \diamond f\|$ once divergence and convergence vertices are known.

8 Concluding remarks

From the *triple path lemma* (Lemma 1), we can see:

Remark 1. We can obtain a DSO with the same space and query time if all interval admission functions of the form $\|st \diamond [u, v]\|$ are replaced by corresponding bottleneck admission functions $\max_{f\in[u,v]}\{\|st \diamond f\|\}$.

So far we have devised $\langle O(n^2), O(1)\rangle$ DSOs. Clearly both of the space complexity and query efficiency have reached asymptotic optima; also its preprocessing time (for the bottleneck admission functions form) is $\tilde{O}(mn)$ (see full version of this paper), which is nearly optimal.

References

1. Abraham, I., Chechik, S., Gavoille, C.: 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. pp. 1199–1218. STOC '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2213977.2214084
2. Abraham, I., Chechik, S., Krinninger, S.: Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In: Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 440–452. SIAM (2017)
3. Abraham, I., Chechik, S., Talwar, K.: Fully dynamic all-pairs shortest paths: Breaking the $o(n)$ barrier. In: APPROX-RANDOM. pp. 1–16 (2014)
4. Arlazarov, V.L., Dinic, E.A., Kronrod, M.A., Faradžev, I.A.: On economical construction of the transitive closure of a directed graph. Soviet Mathematics—Doklady 11(5), 1209–1210 (1970)
5. Bender, M.A., Farach-Colton, M.: The level ancestor problem simplified. Theoretical Computer Science 321(1), 5–12 (2004)
6. Bernstein, A., Karger, D.: Improved distance sensitivity oracles via random sampling. In: Proceedings 19th ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 34–43 (2008)
7. Bernstein, A., Karger, D.: A nearly optimal oracle for avoiding failed vertices and edges. In: Proceedings 41st Annual ACM Symposium on Theory of Computing (STOC). pp. 101–110 (2009)

8. Chechik, S., Langberg, M., Peleg, D., Roditty, L.: f-sensitivity distance oracles and routing schemes. Algorithmica 63(4), 861–882 (Aug 2012), http://dx.doi.org/10.1007/s00453-011-9543-0
9. Demetrescu, C., Italiano, G.F.: A new approach to dynamic all pairs shortest paths. J. ACM 51(6), 968–992 (2004)
10. Demetrescu, C., Thorup, M., Chowdhury, R.A., Ramachandran, V.: Oracles for distances avoiding a failed node or link. SIAM J. Comput. 37(5), 1299–1318 (2008)
11. Demetrescu, C., Italiano, G.F.: Fully dynamic all pairs shortest paths with real edge weights. J. Comput. Syst. Sci. 72(5), 813–837 (Aug 2006), http://dx.doi.org/10.1016/j.jcss.2005.05.005
12. Demetrescu, C., Thorup, M.: Oracles for distances avoiding a link-failure. In: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 838–843. SODA '02, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2002), http://dl.acm.org/citation.cfm?id=545381.545490
13. Duan, R., Pettie, S.: Dual-failure distance and connectivity oracles. In: Proceedings 20th ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 506–515 (2009)
14. Fredman, M.L., Willard, D.E.: Surpassing the information-theoretic bound with fusion trees. J. Comput. Syst. Sci. 47(3), 424–436 (1993)
15. Grandoni, F., Williams, V.V.: Improved distance sensitivity oracles via fast single-source replacement paths. In: FOCS. pp. 748–757. IEEE Computer Society (2012), http://dblp.uni-trier.de/db/conf/focs/focs2012.htmlGrandoniW12
16. Henzinger, M., Krinninger, S., Nanongkai, D.: Dynamic approximate all-pairs shortest paths: Breaking the o(mn) barrier and derandomization. In: FOCS 2013 54th Annual IEEE Symposium on Foundations of Computer Science. pp. 538–547. Proceedings 2013 IEEE 54th Annual Symposium on Foundations of Computer Science FOCS 2013, IEEE, Los Alamitos, CA (October 2013), http://eprints.cs.univie.ac.at/3747/
17. Henzinger, M., Krinninger, S., Nanongkai, D.: Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In: 46th ACM Symposium on Theory of Computing (STOC 2014) (June 2014), http://eprints.cs.univie.ac.at/4042/
18. Henzinger, M., Krinninger, S., Nanongkai, D.: A subquadratic-time algorithm for decremental single-source shortest paths. In: SODA 2014. SIAM, Philadelphia (January 2014), http://eprints.cs.univie.ac.at/3785/
19. Hershberger, J., Suri, S.: Vickrey prices and shortest paths: what is an edge worth? In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science (FOCS). pp. 252–259 (2001), erratum, Proc. 43rd FOCS, p. 809, 2002
20. Khanna, N., Baswana, S.: Approximate shortest paths avoiding a failed vertex: Optimal size data structures for unweighted graphs. In: 27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France. pp. 513–524 (2010), http://dx.doi.org/10.4230/LIPIcs.STACS.2010.2481
21. Patrascu, M., Roditty, L.: Distance oracles beyond the thorup-zwick bound. In: Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science. pp. 815–823. FOCS '10, IEEE Computer Society, Washington, DC, USA (2010), http://dx.doi.org/10.1109/FOCS.2010.83
22. Thorup, M.: Worst-case update times for fully-dynamic all-pairs shortest paths. In: Proceedings 37th ACM Symposium on Theory of Computing (STOC). pp. 112–119 (2005)