Distributed Distance Sensitivity Oracles

Vignesh Manoharan(✉) and Vijaya Ramachandran

The University of Texas at Austin, Austin, TX, USA
{vigneshm,vlr}@cs.utexas.edu

Abstract. We present results for the distance sensitivity oracle (DSO) problem, where one needs to preprocess a given directed weighted graph $G = (V, E)$ in order to answer queries about the shortest path distance from s to t in G that avoids edge e, for any $s, t \in V, e \in E$. No non-trivial results are known for DSO in the distributed CONGEST model even though it is of importance to maintain efficient communication under an edge failure. We present DSO algorithms with different tradeoffs between preprocessing and query cost – one that optimizes query response rounds, and another that prioritizes preprocessing rounds. We complement these algorithms with unconditional CONGEST lower bounds. Additionally, we present almost-optimal upper and lower bounds for the related all pairs second simple shortest path (2-APSiSP) problem.

Keywords: Graph Algorithms · Shortest Paths

1 Introduction

In distributed networks, maintaining communication in the event of a link failure is an important problem. In a communication network modeled by a graph $G = (V, E)$, consider the failure of some edge $e \in E$. We investigate the problem of computing shortest path distance when such an edge fault occurs. This is a fundamental problem in networks for routing and communication. Specifically, we need to answer queries of the form $d(s, t, e)$ for $s, t \in V, e \in E$, where $d(s, t, e)$ is the shortest path distance from s to t when edge e is removed from G. In the special case when s and t are fixed vertices this is known as the replacement paths (RPaths) problem.

Let $|V| = n, |E| = m$. A naive algorithm could compute all n^2m distances $d(s, t, e)$ (this can be reduced to $O(n^3)$ distances since we are only interested in edges on the s-t shortest path), but such an algorithm would have high complexity ($\Omega(n^2)$ rounds in CONGEST) simply due to the output size. So, we instead consider Distance Sensitivity Oracles (DSO), which have been studied in sequential setting. In DSO we first preprocess the network G and store certain information about distances. Then, we answer query $d(s, t, e)$ using the stored information. Without any preprocessing, we can answer a query using a single source shortest path (SSSP) computation.

This work was supported in part by NSF grant CCF-2008241.

U. Schmid and R. Kuznets (Eds.): SIROCCO 2025, LNCS 15671, pp. 366–383, 2025.
https://doi.org/10.1007/978-3-031-91736-3_22

In the sequential setting, the first algorithm for constructing a DSO was given in [8], which was improved in [4] to obtain an algorithm with $O(1)$ query time and $\tilde{O}(mn)$ preprocessing time, which matches the sequential APSP running time. Despite the importance of DSO especially in the distributed setting, no CONGEST results were known.

In this paper, we present the first algorithms to construct a DSO tailored to the CONGEST model. We present two algorithms – one that prioritizes preprocessing rounds, and another that obtains efficient queries at the cost of higher preprocessing. We complement our algorithms with unconditional lower bounds on the tradeoff between preprocessing and query rounds. Adapting this technique, we present a CONGEST lower bound for k-source SSSP – this appears to be the first CONGEST lower bound between n and $\sqrt{n}$ for a shortest path problem.

Another problem we consider is the closely related all pairs second simple shortest paths problem (2-APSiSP), where instead of answering queries $d(s,t,e)$ when a particular edge e fails, we are only interested in a simple s-t path that differs from a shortest path in G by at least one edge. The shortest such path is called a second simple shortest path (2-SiSP) [23], and we denote its distance by $d_2(s,t)$. In the 2-APSiSP problem, we need to compute $d_2(s,t)$ for all pairs $s,t \in V$. This problem was studied in the sequential setting in [1]. In this paper we present a CONGEST algorithm to compute 2-APSiSP in $\tilde{O}(n)$ rounds and we present a matching lower bound that applies even when APSP distances in the original graph are known beforehand. Unlike the DSO problem, there is no need for separate preprocessing and query response rounds.

A core procedure used in sequential DSO and 2-APSiSP algorithms [1,4,8] is that of computing excluded shortest paths distances, where we need to compute shortest path distances when certain types of paths are avoided one at a time. We present a distributed CONGEST procedure for this computation which we use in our 2-APSiSP algorithm and our first DSO algorithm.

Roadmap. The rest of this paper is organized as follows. After presenting background information and problem definitions in Sect. 1.1 we describe our results in Sect. 1.2 and prior work in Sect. 1.3. We present our DSO algorithms and lower bounds in Sect. 2 and our near-optimal 2-APSiSP upper and lower bounds in Sect. 3. Some proof details are deferred to the full version of this paper [16].

1.1 Preliminaries

The CONGEST Model. In the CONGEST model [19], a communication network is represented by a graph $G = (V, E)$ where nodes model processors and edges model bounded-bandwidth communication links between processors. Each node has a unique identifier in $\{0, 1, \ldots n-1\}$ where $n = |V|$, and each node only knows the identifiers of itself and its neighbors in the network. Each node has infinite computational power. The nodes perform computation in synchronous rounds, where each node can send a message of up to $\Theta(\log n)$ bits to each neighbor and can receive the messages sent to it by its neighbors. The complexity of an algorithm is measured by the number of rounds until the algorithm terminates.

We mainly consider directed weighted graphs G in this paper, where each edge has an integer weight known to the vertices incident to the edge. Following the convention for CONGEST algorithms [6,18], the communication links are always bi-directional and unweighted.

Notation and Terminology. We will deal primarily with directed weighted graphs $G = (V, E)$. Let $|V| = n$. Each edge $(s,t) \in E$ (for $s,t \in V$) can have non-negative integer weight $w(s,t)$, where $W = poly(n)$. We denote the shortest path distance from s to t by $d(s,t)$, and a shortest path from s to t by P_{st}. The undirected diameter of G, denoted D, is the maximum shortest path distance between two vertices in the underlying undirected unweighted graph of G. We use $G^r = (V, E^r)$ to denote the reversed graph of G, where each edge is oriented in the opposite direction.

We use $d(s,t,e)$, for $s,t \in V, e \in E$, to denote the shortest path distance from s to t in the graph $G - \{e\}$, i.e., the graph G with edge e removed, also called replacement path distance. We generalize this definition for a path P, and $d(s,t,P)$ denotes the shortest path distance from s to t in $G - E(P)$. We use $d_2(s,t)$ to denote the second simple shortest path distance from s to t (2-SiSP distance), which is the minimum distance of a simple s-t path that differs from an s-t shortest path P_{st} by at least one edge. Note that $d_2(s,t) = \min_{e \in P_{st}} d(s,t,e)$. The distance $d_2(s,t)$ is independent of the chosen shortest path P_{st}, and $d_2(s,t) = d(s,t)$ if there are multiple s-t shortest paths.

Shortest Path Trees and Independent Paths: We denote the out-shortest path tree rooted at a vertex $x \in V$ by T_x. For a subpath P of T_x, we define $T_x(P)$ to be the subtree of T_x rooted at the vertex in P *furthest* from source x. We define the notion of a set of *independent* paths $\mathcal{R}$ with respect to a shortest path tree T_x as follows: Each path $P \in \mathcal{R}$ is a subpath of T_x, and for any pair of paths $P, P' \in \mathcal{R}$, for any two vertices $u \in P, u' \in P'$, the subtree of T_x rooted at u and the subtree of T_x rooted at u' are disjoint.

Scheduling with Random Delays: Following [9,12], we use scheduling with random delays in order to efficiently perform multiple computations on the network. Let the *congestion* of a CONGEST algorithm be the maximum number of messages it sends through a single edge, over all rounds of the algorithm. Consider an algorithm that runs in R rounds and has congestion C. Then, we can run k instances of the algorithm in $O(kC + R\log n)$ rounds, with total congestion $O(kC)$ over all instances.

We use basic CONGEST primitives such as breadth-first search (BFS) [13], broadcast and convergecast [19] in our algorithms. We use $SSSP$ and $APSP$ to denote the round complexity in the CONGEST model for weighted single source shortest paths (SSSP) and weighted all pairs shortest paths (APSP) respectively. The current best algorithm for weighted APSP runs in $\tilde{O}(n)$ rounds, randomized [5]. For weighted SSSP, [6] provides an $\tilde{O}(\sqrt{n} + n^{2/5+o(1)} D^{2/5} + D)$ round randomized algorithm. The current best lower bounds are $\tilde{\Omega}\left(\sqrt{n} + D\right)$ for weighted SSSP [20,21] and $\tilde{\Omega}\left(n\right)$ for (weighted and unweighted) APSP [18]. In our algorithms, we also use a low congestion SSSP procedure presented in [11], which takes $\tilde{O}(n)$ rounds but incurs only $\tilde{O}(1)$ congestion per edge. When computing shortest path distances from a source x, we assume that each vertex v knows the preceding and succeeding vertices on a shortest path from x to v. This can be achieved using $O(1)$ rounds per source, by each vertex sharing its distance from x with its neighbors, e.g. y precedes v on the x-v shortest path if $d(x, v) = d(x, y) + w(y, v)$.

Problem Definitions. In the following definitions, $G = (V, E)$ is a directed weighted graph.

Excluded Shortest Paths: Given a set of sources $X \subseteq V$ and sets of independent paths $\mathcal{R}_x$ for each $x \in X$ (as defined in Sect. 1.1), we need to compute $d(x, y, P)$ for each $x \in X, y \in V$ and $P \in \mathcal{R}_x$, i.e., the x-y shortest path distance when path P is removed. The input set of paths $\mathcal{R}_x$ is given as follows: Each edge $e \in E$ that belongs to some path $P \in \mathcal{R}_x$ knows the start vertex of P. For the output, each distance $d(x, y, P)$ must be known at node y.

Distance Sensitivity Oracles (DSO): We need to compute $d(x, y, e)$, the x-y shortest path distance when e is removed, for any pair of vertices $x, y \in V$ and $e \in E$. A DSO algorithm is allowed to perform some preprocessing on the graph G, and then answer queries of the form $d(x, y, e)$. We assume that at query time the query $d(x, y, e)$ is announced to every vertex in the graph. The round complexities of our algorithms are unchanged if we instead assume the query is known to only one node, since we can broadcast the query to all nodes in D rounds once it is known (and all our query bounds are $\Omega(D)$). After the query is answered, distance $d(x, y, e)$ must be known at node y.

All Pairs Second Simple Shortest Path (2-APSiSP): We need to compute $d_2(x, y)$, the x-y second simple shortest path distance (2-SiSP distance, see Sect. 1.1), for all pairs of vertices $x, y \in V$. For the output, each distance $d_2(x, y)$ $(\forall x \in V)$ must be known at node y. Unlike DSO, there are no separate rounds for preprocessing and answering queries.

1.2 Results

We summarize our results for DSO and 2-APSiSP in a directed weighted graph $G = (V, E)$ in the CONGEST model. Recall that $n = |V|$, and D is the undirected diameter of G. All our algorithms are randomized and are correct w.h.p. in n (with probability $1 - (1/poly(n))$).

Excluded Shortest Paths. Our first DSO algorithm and our 2-APSiSP algorithm rely on a subroutine to efficiently compute shortest path distances from multiple sources when certain parts of shortest path trees are excluded, one at a time. We present an algorithm for the excluded shortest paths problem defined in Sect. 1.1. Our CONGEST implementation is inspired by a sequential algorithm for a single source in [8], and we extend the result to multiple sources by using low congestion SSSP combined with efficient random scheduling. We defer the proof of our result stated in Theorem 1 to the full version of this paper [16].

Theorem 1. *Given a set of sources $X \subseteq V$ and an independent set of paths $\mathcal{R}_x$ for each source $x \in X$, we can compute $d(x, y, P)$ for each $x \in X, y \in V, P \in R_x$ in $\tilde{O}(n)$ rounds. Additionally, the maximum congestion is $\tilde{O}(|X|)$.*

Distance Sensitivity Oracles (DSOs). We present two CONGEST algorithms for DSO, one with fast query responses and one with fast preprocessing. We complement these algorithms with unconditional lower bounds on preprocessing rounds. Our DSO algorithms focus on computing the distance value of a given query $d(s, t, e)$, but we can also readily augment our algorithms so that each vertex on the replacement path from x to y avoiding e knows the next vertex on that path.

DSO with Fast Query Responses. Our first DSO optimizes query time, answering any query $d(s, t, e)$ in $O(D)$ rounds. More specifically, the DSO uses a constant number of broadcasts to answer a query, after which all nodes know $d(s, t, e)$. To answer a batch of k different queries, we pipeline the broadcasts for each query, answering them all in $O(k + D)$ rounds. This DSO uses $\tilde{O}(n^{3/2})$ rounds for preprocessing, and is described in Sect. 2.1.

The algorithm performs $\tilde{O}(\sqrt{n})$ exclude computations from every vertex, such that any query $d(s, t, e)$ can be answered using $O(1)$ excluded shortest paths distances stored at different vertices. We use two types of exclude computations: (i) to directly compute $d(s, t, e)$ when edge e is within $\sqrt{n}$ hops of s or t, and (ii) to exclude independent segments of each out-shortest path tree T_s, and combine appropriate excluded shortest path distances to compute $d(s, t, e)$ if e is not within $\sqrt{n}$ hops of s or t. Once the query is known, we broadcast the relevant precomputed distances and compute $d(s, t, e)$ at t in $O(D)$ rounds.

Theorem 2. *We can construct a DSO for directed weighted graphs that takes $\tilde{O}(n^{3/2})$ preprocessing rounds, and then answers a batch of any k queries of the form $d(s, t, e)$ in $O(k + D)$ rounds.*

DSO with Fast Preprocessing. We present a DSO construction that prioritizes preprocessing time, and takes $\tilde{O}(n)$ rounds. Each query $d(s, t, e)$ takes $\tilde{O}(\sqrt{n} + D)$ rounds to answer. To answer a batch of queries, we pipeline part of the query procedure, answering k queries in $\tilde{O}(k\sqrt{n} + D)$ rounds. Note that answering k queries $d(s, t, e)$ without any preprocessing can be done trivially with k SSSP computations in $\tilde{O}(k \cdot (\sqrt{n} + n^{2/5}D^{2/5} + D))$ rounds [6], and we improve on this bound for all $k \geq 1$. We describe this algorithm in Sect. 2.2.

In order to reduce preprocessing rounds to $\tilde{O}(n)$, we cannot perform too many exclude computations. So, we handle short replacement paths of hop-length $\leq \sqrt{n}$ at query time using a $\sqrt{n}$-hop Bellman-Ford procedure. For longer replacement paths ($> \sqrt{n}$ hops), we construct randomly sampled graphs where each edge is included with certain probability (inspired by a sequential DSO in [22]). In these graphs, we perform SSSP computations only from certain sampled vertices during preprocessing. This allows us to quickly compute long replacement path distances at query time.

Theorem 3. *We can construct a DSO for directed weighted graphs that takes $\tilde{O}(n)$ preprocessing rounds, and then answers a batch of any k queries of the form $d(s,t,e)$ in $\tilde{O}(k\sqrt{n}+D)$ rounds.*

DSO Lower Bounds. We show a lower bound on the preprocessing rounds used by a DSO that achieves a given query response round complexity. We consider algorithms that answer a batch of k queries in $O(k \cdot Q)$ rounds where $Q = o(\sqrt{n}/\log n)$. This covers the salient range of query response complexities, since we can use SSSP to answer queries with no preprocessing and the current best CONGEST lower bound for SSSP is $\Omega(\sqrt{n}+D)$. Our lower bound applies even for directed unweighted graphs with diameter $\log n$.

Theorem 4. *Consider a DSO algorithm for directed unweighted graph $G = (V,E)$ on n nodes with diameter $D = \Omega(\log n)$ that performs P rounds of preprocessing and then answers any batch of $k \leq \frac{n}{Q^2 \log^2 n}$ queries in $O(k \cdot Q + D)$ rounds, where $Q = o\left(\frac{\sqrt{n}}{\log n}\right)$. Then, P is $\tilde{\Omega}\left(\frac{n}{Q}\right)$.*

For the setup in Theorem 2, for a DSO algorithm that answers k queries in $O(k+D)$ rounds, we show a lower bound of $\tilde{\Omega}(n)$ preprocessing rounds. Complementing Theorem 3, we show a lower bound of $\tilde{\Omega}(\sqrt{n})$ preprocessing rounds for a DSO algorithm that answers k queries in $o\left(k\frac{\sqrt{n}}{\log n}+D\right)$ rounds. We present these lower bounds for directed unweighted graphs and they trivially apply to directed weighted graphs as well.

Our main technique to obtain this lower bound is to adapt a CONGEST lower bound for SSSP given in [21]. We adapt their construction to k sources for any $1 \leq k \leq \frac{n}{2}$, obtaining a lower bound of $\tilde{\Omega}(\sqrt{nk})$ for the problem of computing shortest path distances from k sources (k-source SSSP) in directed unweighted graphs, stated in Theorem 5. We note that CONGEST algorithms for k-source SSSP have been studied in [14], and our results show that the round complexity $\tilde{O}(\sqrt{nk}+D)$ in [14] is almost optimal for $k \geq n^{1/3}$.

Theorem 5. *Given a directed unweighted graph $G = (V,E)$ with diameter $D = \Omega(\log n)$, computing k-source SSSP requires $\tilde{\Omega}(\sqrt{nk}+D)$ rounds.*

All Pairs Second Simple Shortest Paths (2-APSiSP). We present an algorithm for 2-APSiSP that takes $\tilde{O}(n)$ rounds to compute the second simple shortest path distance $d_2(s,t)$ for all pairs of vertices $s,t \in V$. Our algorithm builds on a characterization of the second simple shortest path used in a sequential algorithm [1]. We use one exclude computation from each vertex, followed by non-trivial local computation in order to compute all 2-APSiSP distances.

Theorem 6. *We can compute directed weighted 2-APSiSP in $\tilde{O}(n)$ rounds.*

2-APSiSP Lower Bound. We present a lower bound of $\tilde{\Omega}(n)$ for computing 2-APSiSP in undirected unweighted graphs, which proves that our $\tilde{O}(n)$ round algorithm is almost optimal for any class of graphs: directed or undirected, weighted or unweighted. An $\tilde{\Omega}(n)$ CONGEST lower bound for computing directed weighted 2-SiSP for one pair was shown in [15] and trivially applies to 2-APSiSP as well. But our lower bound for 2-APSiSP is stronger as the bound in [15] (for 2-SiSP) applies only to directed weighted graphs and requires the input pair of vertices to have a $\Theta(n)$-hop shortest path. Our lower bound applies even when APSP distances are known, and in constant diameter graphs. We present our result in Sect. 3.

Theorem 7. *Given an undirected unweighted graph $G = (V,E)$, computing 2-APSiSP requires $\tilde{\Omega}(n)$ rounds, even if APSP distances are known (i.e. vertex t knows distances $d(s,t)$ $\forall s \in V$). This lower bound applies even for graphs with constant diameter, and against $(3-\epsilon)$-approximation algorithms (for any constant $\epsilon > 0$). In directed graphs, it holds against any approximation ratio.*

1.3 Prior Work

Distance Sensitivity Oracle. The DSO problem has been studied extensively in the sequential setting (e.g. [4,8,22]). No non-trivial algorithms for DSO have been studied in the distributed CONGEST setting, but some related problems have received attention. For computing replacement path distances between a single pair of vertices (RPaths), CONGEST algorithms and lower bounds were presented in [15] for directed and undirected, weighted and unweighted graphs with almost optimal bounds in most cases: these include a $\tilde{\Theta}(n)$ bound (tight up to polylog factors) for directed weighted graphs, and bounds almost matching *SSSP* round complexity for undirected graphs. Very recently, matching CONGEST upper and lower bounds for RPaths in directed unweighted graphs were given in [7]. A distributed $O(D \log n)$ algorithm for single source replacement paths in undirected unweighted graphs was given in [10].

All Pairs Second Simple Shortest Paths. For a single pair of vertices, an $\tilde{O}(mn)$ sequential time algorithm for the 2-SiSP problem in directed weighted graphs was given in [23]. For 2-APSiSP, a sequential $\tilde{O}(mn)$ time algorithm matching the 2-SiSP running time was given in [1]. A fine-grained sequential lower bound of $\Omega(mn)$ complementing these upper bounds was shown in [2] for directed weighted 2-SiSP. In the distributed setting, the problem of computing 2-SiSP for a single pair has been studied in [15] with both upper and lower bounds: these include an almost optimal $\tilde{\Theta}(n)$ bound for directed weighted graphs and a $\Theta(SSSP)$ bound for undirected graphs.

2 Distance Sensitivity Oracles

In the DSO problem, we need to preprocess an input graph $G = (V, E)$ so that we can answer query $d(x, y, e)$ for any pair of vertices $x, y \in V$, edge $e \in E$ to be avoided. We present algorithms for directed weighted graphs, though they trivially also apply to undirected or unweighted graphs. Note that with no preprocessing, a query can be answered in $O(SSSP) = \tilde{O}(\sqrt{n} + n^{2/5+o(1)}D^{2/5} + D)$ rounds [6]. We present two algorithms that beat this bound in Sects. 2.1 and 2.2. We complement these algorithms with lower bounds in Sect. 2.3.

In both our algorithms we assume that e is on a x-y shortest path. As both algorithms first compute APSP distances, we can readily check if $e = (z, z')$ is on a x-y shortest path by checking if $d(x, y) = d(x, z) + w(e) + d(z', y)$, and return $d(x, y, e) = d(x, y)$ if not.

2.1 DSO with Fast Query Responses

Our first method for directed weighted DSO with fast query responses is described in Algorithm 1. Our algorithm builds on a sequential DSO algorithm in [8] that runs in time $O(mn^{1.5} + n^{2.5}\log n)$ and answers queries in $O(1)$ time. The main idea is to consider replacement distances $d(x, y, e)$ of two forms in the preprocessing phase. If e is within $\sqrt{n}$ hops of x in its shortest path tree T_x, we perform exclude computations from each node, excluding edges within $\sqrt{n}$ depth in its outgoing shortest path tree. We do the same on the reversed graph.

Tree Cutting Procedure. To compute $d(x, y, e)$ when edge e is at least $\sqrt{n}$ hops away from both x and y, we make use of interval computations. We use a *tree cutting procedure* on each shortest path tree T_x, which designates certain vertices as *level vertices* such that the subpaths of T_x in the intervals between level vertices can be arranged into $O(\sqrt{n})$ sets of independent paths. We ensure that any edge e that is $\sqrt{n}$ hops away from both x and y is flanked by some pair of level vertices w.h.p. in n, and use the distances when these intervals are excluded to answer query $d(x, y, e)$ as shown below.

In order to identify appropriate level vertices, we use the following method from [8]: In shortest path tree T_x, let the size of vertex y be the number of nodes in the subtree of T_x rooted at y. If a vertex y has more than one child with size $\geq \sqrt{n}$, all vertices at the same depth as y in T_x are designated *type 1* level vertices. As noted in [8] there are at most $\sqrt{n}$ such y. We are only concerned with $\sqrt{n}$-length paths, so we do not need to consider nodes of size $< \sqrt{n}$ when computing *type 1* level vertices. Additionally, all vertices at depth $i\sqrt{n}$ in T_x, for $i = 1, 2, \cdots, \sqrt{n}$ are designated *type 2* level vertices so that any path of length $\geq \sqrt{n}$ contains a level vertex. Specifically, if edge e on a x-y shortest path is $\sqrt{n}$ hops away from both x and y, e is flanked by some pair of level vertices on the x-y path. Combining both types, level vertices occur at $\leq 2\sqrt{n}$ depth values, and vertices at these depths partition the tree into $O(\sqrt{n})$ intervals. The set of paths in a given interval with level vertex endpoints on either side are independent

Algorithm 1 DSO with fast query: $\tilde{O}(n^{3/2})$ preprocessing and $\tilde{O}(D)$ query

```
Input: Graph $G = (V, E)$.
Output: Answer query for replacement path distance $d(x, y, e)$ for any $x, y \in V, e \in E$.
 1: ▷ Preprocessing Algorithm ◁
 2: Compute APSP on $G$ and the reversed graph $G^r$, remembering parent nodes.
 3: Perform excluded shortest path computations from each vertex $x \in V$, with sets of
    edges $R_x^{(1)}, \cdots, R_x^{(\sqrt{n})}$ where $R_x^{(i)}$ is the set of edges $i$ hops from $x$ in the out-shortest
    path tree $T_x$ rooted at $x$. Repeat in $G^r$.
 4: for $x \in V$ do
 5:     ▷ Tree cutting procedure on $T_x$ ◁
 6:     Perform an upcast along $T_x$, computing size and depth of each vertex, and
        identify level vertices of both types. Perform an additional upcast and downcast,
        to identify all type 1 level vertices. (details described in Section 2.1)
 7:     Perform a downcast along $T_x$ so that each edge $e \in T_x$ knows the closest level
        vertex (if there is one) on the path from $x$ to $e$. Repeat in $G^r$. Each edge
        identifies the subpath between interval vertices it belongs to.
 8:     Perform exclude computations from $x$ with the $O(\sqrt{n})$ sets of independent paths
        identified in line 7.
 9: ▷ Query Algorithm: Given query $d(x, y, e)$ ◁
10: ▷ $e$ is within $h$ hops from $x$ or $y$ in $T_x$: use excluded distances from line 3 ◁
11: If $d(x, y, e)$ has been computed in line 3 at $x$ or $y$, broadcast $d(x, y, e)$.
12: ▷ $e$ is not within $h$ hops from $x$ or $y$ in $T_x$: use excluded distances from line 8 ◁
13: From line 7, $e$ locally determines $a$ as the closest level vertex on the path from $x$
    to $e$, and $b$ as the closest level vertex on the path from $e$ to $y$. $e$ broadcasts the
    identities of $a$ and $b$.
14: Then, $a$ broadcasts $d(a, y, e)$ and $x$ broadcasts $d(x, a), d(x, b, e)$. Distances $d(a, y, e)$
    and $d(x, b, e)$ were computed in line 3.
15: Finally, $y$ broadcasts $d(x, y, e) = \min(d(x, a) + d(a, y, e), d(x, b, e) +$
    $d(b, y), d(x, y, [a, b]))$. Distance $d(x, y, [a, b])$ was computed in line 8.
```

(as shown in [8]). So, we have arranged the subpaths of the tree between level vertices into $O(\sqrt{n})$ sets of independent paths.

In lines 6–8 of Algorithm 1, we implement this method as follows: In line 6, we perform an upcast along T_x in $O(n)$ rounds, so that each vertex knows its size and depth in T_x, and use this to identify level vertices of both types described in the paragraph above. With an additional upcast and downcast, all vertices at the same depth as some type 1 level vertex also identifies itself as a level vertex. After the downcast in line 7, each edge knows its closest level vertex in either direction. So, in the exclude computation in line 8, each edge knows which interval subpath it is a part of. We use $O(n)$ rounds and $O(1)$ congestion per tree T_x for the computation in lines 6, 7.

Analysis of Algorithm 1. We utilize the multiple excluded shortest paths algorithm from Theorem 1 for efficiently excluding paths in shortest path trees. We perform a total of $\sqrt{n}$ exclude computations per source x in lines 3, 8, which takes $\tilde{O}(n^{3/2})$ rounds using Theorem 1. In the query algorithm, given query $d(x, y, e)$, we identify the level vertices flanking e if they exist in line 13. We broadcast the excluded shortest paths distances necessary to compute $d(x, y, e)$ in lines 11, 14 in $O(D)$ rounds, and compute the result at y.

Proof (of Theorem 2). **Correctness:** Let P_{xy} be a replacement path from x to y when e is removed. If e is within $\sqrt{n}$ hops from x or y in the shortest path tree, then $d(x, y, e)$ is computed in line 3 and the query is answered in line 11.

Now, consider the other case where e is not within h hops of x or y on the x-y shortest path. After the tree cutting procedure on T_x, there are level vertices a, b such that a is on the shortest path from x to e and b is on the shortest path from e to y. Additionally, the interval subpath from a to b has at most $\sqrt{n}$ hops, so e is within $\sqrt{n}$ hops of both a and b in their respective shortest path trees. Thus, $d(a, y, e)$ is computed at y in line 3 and similarly $d(x, b, e)$ is computed at x in the reversed graph. After the identities of a, b are broadcast, the replacement path distance is correctly computed in line 15—the first term is correct when P_{xy} passes through a and the second is correct when it passes through b. If P_{xy} passes through neither a nor b, it skips the whole interval $[a, b]$, and the distance is correctly computed as $d(x, y, [a, b])$.

Round Complexity: Computing APSP takes $\tilde{O}(n)$ rounds. We perform $\tilde{O}(n^{3/2})$ different exclude computations in lines 3 and 8. Using Theorem 1, this takes $\tilde{O}(n^{3/2})$ rounds. The downcast in line 7 takes $O(n)$ rounds and $O(1)$ congestion per vertex, and can be scheduled for all x in $\tilde{O}(n)$ rounds. The query algorithm broadcasts $O(1)$ values and then broadcasts the correct answer, so it takes $O(D)$ rounds with $O(1)$ congestion per edge. Thus, we can answer a batch of k queries by pipelining all broadcasts in $O(k + D)$ rounds.

2.2 DSO with Fast Preprocessing

In this section, we present Algorithm 2 to construct a DSO with $\tilde{O}(n)$ rounds for preprocessing, which answers a batch of k queries in $\tilde{O}(k \cdot \sqrt{n} + D)$ rounds. The algorithm uses a graph sampling approach to compute hop-limited replacement paths, inspired by a sequential DSO construction [22]. We first describe this graph sampling method and then describe our distributed algorithm which uses it along with vertex sampling to achieve efficient preprocessing.

Hop-Limited Replacement Paths. Given a hop parameter h, and a graph $G = (V, E)$ along with a set of sources $S \subseteq V$, we wish to compute h-hop replacement path distances $d_h(s, x, e)$ for $s \in S, x \in V, e \in E$. Here, $d_h(s, x, e)$ denotes the minimum weight of an s-x path containing up to h edges that does not contain e. The following lemma can be established by adapting proofs in [22], see the full version [16] for details.

Lemma 1. *Sample* $\tilde{h} = 15h \log n$ *graphs* $G_1, \cdots, G_{\tilde{h}}$ *as subgraphs of* G *by including each edge of* G *with probability* $1 - \frac{1}{h}$. *Then,*

A. W.h.p. in n, *there are* $\Theta(\log n)$ *graphs* G_i *such that* $e \notin G_i$.

Algorithm 2 DSO with fast preprocessing: $\tilde{O}(n)$ preprocessing and $\tilde{O}(\sqrt{n}+D)$ query

```
Input: Graph $G = (V, E)$.
Output: Answer query for replacement path distance $d(x, y, e)$ for any $x, y \in V, e \in E$.
 1: ▷ Preprocessing Algorithm ◁
 2: for $j = \log(\sqrt{n}), \cdots, \log n$ do
 3:     ▷ Preprocessing for replacement paths of hop-length $[2^j, 2^{j+1}]$, where $2^j \geq \sqrt{n}$ ◁
 4:     Each vertex $s \in V$ is sampled and added to $S_j$ with probability $\frac{c \log n}{2^j}$.
 5:     Let $h = 2^{j+1}$. Sample $\tilde{O}(h)$ graphs $G_i^j$ as in Lemma 1 by randomly sampling
        each edge with probability $1 - (1/h)$. ▷ Each edge locally samples itself
 6:     For each $e \in G$, $I_e^j \leftarrow \{(j, i) \mid e \notin G_i^j\}$ ▷ Computed locally at endpoints of $e$
 7:     Compute SSSP from each vertex in $S_j$, on each $G_i^j$ (and reversed graph). The
        distance in graph $G_i^j$ is denoted $d_i^j$. Distances $d_i^j(s, x), d_i^j(x, s)$ are computed at
        $x$ for each $s \in S_j, x \in V$.
 8: ▷ Query Algorithm: Given query $d(x, y, e)$ ◁
 9: ▷ Find best replacement path of hop-length $\leq \sqrt{n}$ ◁
10: Perform a distributed Bellman-Ford computation in $G - \{e\}$ starting from $x$, up
    to $\sqrt{n}$ hops. This computes $d_{\sqrt{n}}(x, y, e)$ at $y$.
11: ▷ Find best replacement path of hop-length $> \sqrt{n}$ ◁
12: for $j = \log(\sqrt{n}), \cdots, \log n$ do
13:     ▷ Handle paths of hop-length $[2^j, 2^{j+1}]$ ◁
14:     Endpoint of $e$ broadcasts $I_e^j$.
15:     $x$ broadcasts $\{d_i^j(x, s) \mid (j, i) \in I_e^j, s \in S_j\}$, i.e. its distance to sampled vertices
        in $S_j$ in graphs where $e$ is not present.
16:     for $s \in S_j$ do ▷ Local computation at $y$
17:         ▷ $d_s$ is the best $x$-$y$ replacement path distance through $s$ ◁
18:         Compute $d_s = \min_{(j,i) \in I_e^j} d_i^j(x, s) + d_i^j(s, y)$
19: Compute $d(x, y, e) = \min(d_{\sqrt{n}}(x, y, e), \min_{s \in V} d_s)$ ▷ Local computation at $y$
```

B. Let P_{sx}^e be a h-hop replacement path from s to x avoiding e. W.h.p. in n, there is at least one graph G_i such that G_i contains all edges in P_{sx}^e but does not contain e.

CONGEST DSO with Fast Preprocessing. We now describe Algorithm 2. To obtain fast preprocessing, we only deal with replacement paths of hop-length $\geq \sqrt{n}$ during preprocessing. Shorter replacement paths are computed at query time using $\sqrt{n}$ steps of Bellman-Ford with the appropriate edge deleted (line 10).

To compute replacement paths of hop-length $[2^j, 2^{j+1}]$, for $j \geq \frac{1}{2} \log n$, we use the result in Lemma 1 with $h = 2^{j+1}$. After sampling graphs in line 4, we compute shortest paths to and from sampled vertices in line 7. With this information, we can compute replacement path distances at query time (lines 14–18) without needing to compute APSP during preprocessing. Although we could use the graph sampling method for $\leq \sqrt{n}$ hop-length replacement paths and stay within the $\tilde{O}(n)$ preprocessing bound, the number of sampled vertices would exceed $\sqrt{n}$ and answering a query by broadcast (line 15) would take $\omega(\sqrt{n})$ rounds.

Proof (of Theorem 3). **Correctness:** For query $d(x, y, e)$, if the x-y replacement path has hop-length $\leq \sqrt{n}$, the distance is correctly computed in line 10 using the Bellman-Ford computation from x up to $\sqrt{n}$ hops. We now address replacement paths of hop-length $[2^j, 2^{j+1}]$, for $2^j \geq \sqrt{n}$.

Fix one such replacement path P from x to y avoiding edge e, with hop length $2^j \leq h(P) < 2^{j+1}$ and weight $w(P)$. In the preprocessing algorithm, consider the j'th iteration of the loop (lines 2–7). Since we sample vertices into S_j with probability $\tilde{O}(1/2^j)$, w.h.p in n there is at least one vertex s from S_j that is on P. By Lemma 1.B at least one of the sampled subgraphs G_i^j contains all edges of P and does not contain e w.h.p. in n. In this particular G_i^j, the distances $d_i^j(s, y)$ and $d_i^j(x, s)$ are computed in line 7 and we have $w(P) = d_i^j(x, s) + d_i^j(s, y)$.

When answering query $d(x, y, e)$, we use the set I_e^j computed in line 7 and broadcast in line 14. In line 18, we only consider shortest path distances in graphs not containing e, so all distances d_s correspond to valid x-y paths not containing e. For the particular G_i^j that contains P and not e, we correctly compute $d(x, y, e) = d_i^j(x, s) + d_i^j(s, y)$ in line 18.

Round Complexity: We first analyze the preprocessing algorithm. Consider an iteration j: lines 4–6 only involve local computation, and line 7 performs SSSP from $|S_j| = \tilde{O}(n/2^j)$ vertices on $\tilde{O}(2^j)$ graphs. Using the $\tilde{O}(n)$-round low congestion SSSP algorithm from [11], the total congestion over all SSSP computations is at most $\tilde{O}(|S_j| \cdot 2^j) = \tilde{O}(n)$. Thus, using random scheduling, line 7 takes $\tilde{O}(n)$ rounds.

Now, to answer a query $d(x, y, e)$, computing Bellman-Ford up to $\sqrt{n}$ hops in line 10 takes $O(\sqrt{n})$ rounds. The set I_e^j with size $O(\log n)$ is broadcast in $\tilde{O}(D)$ rounds in line 14. Vertex x then broadcasts a set of size $|I_e^j| \cdot |S_j| = \tilde{O}(\frac{n}{2^j}) = \tilde{O}(\sqrt{n})$ (since $2^j \geq \sqrt{n}$) in line 15, which takes $\tilde{O}(\sqrt{n} + D)$ rounds. Lines 18, 19 only involve local computation at y, so the total rounds to answer the query is $\tilde{O}(\sqrt{n} + D)$. To answer a batch of k queries, we can broadcast the $\tilde{O}(k\sqrt{n})$ required values in lines 14, 15 in $\tilde{O}(k\sqrt{n} + D)$ rounds, and perform k Bellman-Ford computations sequentially in $k\sqrt{n}$ rounds – totaling $\tilde{O}(k\sqrt{n} + D)$ rounds.

2.3 DSO Lower Bounds

In this section, we show lower bounds on the preprocessing rounds used by any DSO algorithm with a given query response round complexity. Our proof uses a graph construction to obtain a reduction from set disjointness to the problem of computing k DSO queries in a directed unweighted graph, building on a $\tilde{\Omega}(\sqrt{n})$ CONGEST lower bound for SSSP in [21]. Our construction also gives a lower bound for the problem of computing k-source shortest paths. While similar techniques have been used to show $\sqrt{n}$ lower bounds, our construction appears to give the first CONGEST lower bounds between n and $\sqrt{n}$ for natural shortest path problems.

We start our DSO lower bound with the following preliminary lemma.

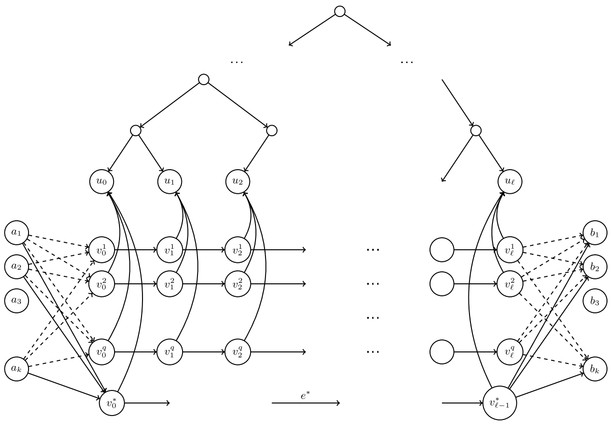

Fig. 1. Lower Bound for DSO Preprocessing Time.

Lemma 2. *Consider an algorithm $\mathcal{A}$ for a directed unweighted graph $G = (V, E)$ on n nodes that computes the answer to any batch of $k \le n/2$ queries $d(a_i, b_i, e_i)$ for $1 \le i \le k$. The answer for query i must be known to b_i after the algorithm terminates. Then, $\mathcal{A}$ must have round complexity $\Omega\left(\frac{\sqrt{nk}}{\log n}\right)$, even on graphs of undirected diameter $\Theta(\log n)$.*

Proof. **Graph Construction:** We construct directed unweighted graph $G = (V, E)$, pictured in Fig. 1, as a balanced binary tree with ℓ leaves, numbered $u_1, \dots u_\ell$, along with a set of q paths v_0^i-v_ℓ^i of length ℓ, and a path v_0^*-$v_{\ell-1}^*$ of length $\ell-1$. The paths are connected to the leaves of the tree as in the figure, with all edges oriented from path to leaf. Additionally, we have $2k$ vertices a_i, b_i for $1 \le i \le k$. We perform a reduction from (kq)-bit set disjointness, and the dashed edges in our construction depend on the inputs S_a, S_b of the set disjointness instance. We index the input such that $S_a[i,j]$ represents the $((i-1)\cdot q + j)$'th bit for $1 \le i \le k, 1 \le j \le q$ (similarly for S_b). The edge (a_i, v_0^j) is present only if $S_a[i,j] = 1$, similarly edge (v_ℓ^j, b_i) is present only if $S_b[i,j] = 1$. This encoding of bits is similar to a lower bound construction in [15]. All other solid edges in Fig. 1 are always present.

Note that $d(a_i, b_i) = \ell + 1$ due to the v_0^*-$v_{\ell-1}^*$ path. When an edge e^* on the v_0^*-$v_{\ell-1}^*$ path is removed, a replacement path from a_i to b_i with weight $\ell + 2$ exists through v_0^j-v_ℓ^j if $S_a[i,j] = S_b[i,j] = 1$ for some j, i.e., $d(a_i, b_i, e^*) = \ell + 2$. Otherwise, either the (a_i, v_0^j) or (v_ℓ^j, b_i) edge is missing for every j, and no a_i-b_i

replacement path exists which means $d(a_i, b_i, e^*) = \infty$. Thus, computing the queries $d(a_i, b_i, e^*)$ for all i allows us to determine set disjointness of S_a, S_b.

Using an analysis similar to ([21], Theorem 3.1), we can prove that if an algorithm $\mathcal{A}$ can compute answer to k queries $d(a_i, b_i, e_i)$ in $R \leq (\ell/2)$ rounds, then we can solve (kq)-bit set disjointness using $Rp \cdot \Theta(\log n)$ bits, where p is the height of tree. See the full version [16] for a detailed proof.

To choose our parameters, we use $p = \Theta(\log n), q = \frac{n-k}{\ell} \geq \frac{n}{2\ell}$ (since $k < n/2$), ℓ will be chosen later. The number of vertices in the graph is $\Theta(\ell \cdot q + k) = \Theta(n)$. The undirected diameter of the graph is $\leq 2p + 4 = \Theta(\log n)$ (using paths through the tree). The number of bits in the set disjointness instance is $kq \geq \frac{kn}{2\ell}$. Using the randomized communication lower bound for set disjointness [3] along with the above claim, we obtain: If $R(n) \leq (\ell/2)$, then $R(n) \cdot \log^2 n = \Omega(kq) \Rightarrow R(n) = \Omega\left(\frac{kn}{\ell \log^2 n}\right)$. Combining the two bounds, $R(n) = \Omega\left(\min(\ell, \frac{kn}{\ell \log^2 n})\right)$. We balance these terms by choosing $\ell = \frac{\sqrt{nk}}{\log n}$, which gives $R(n) = \Omega\left(\frac{\sqrt{nk}}{\log n}\right)$.

Lemma 2 provides a lower bound for computing k replacement path distances in general, and we use it to prove lower bounds for DSOs in terms of preprocessing and query rounds to prove Theorem 4.

Proof (of Theorem 4). Applying Lemma 2, $P + k \cdot Q = \Omega\left(\frac{\sqrt{nk}}{\log n}\right)$. Choosing k such that $k \cdot Q = o\left(\frac{\sqrt{nk}}{\log n}\right)$, would give a lower bound on P. Thus, $\sqrt{k} = o\left(\frac{\sqrt{n}}{Q \log n}\right) \Rightarrow k = o\left(\frac{n}{Q^2 \log^2 n}\right)$. Then, $k \cdot Q + D = \frac{n}{Q \log^2 n} + \Theta(\log n) = o\left(\frac{\sqrt{nk}}{\log n}\right)$. Choosing $k = \Theta\left(\frac{n}{Q^2 \log^3 n}\right)$ for instance, we get $P = \Omega\left(\frac{\sqrt{nk}}{\log n}\right) = \tilde{\Omega}\left(\frac{n}{Q}\right)$.

We now apply Theorem 4 to obtain lower bounds mirroring the upper bounds in Sect. 2, with query setups similar to Algorithm 1 (Corollary 1.A) and Algorithm 2 (Corollary 1.B). Algorithm 2 has query response rounds $\tilde{O}(k\sqrt{n} + D)$, and the query setup in Corollary 1.B differs from it only by a polylog factor.

Corollary 1. *Consider a DSO algorithm $\mathcal{A}$ for directed unweighted graph $G = (V, E)$ on n nodes that uses P rounds for preprocessing.*

A. If $\mathcal{A}$ answers k queries in $O(k + D)$ rounds then P is $\tilde{\Omega}(n)$.
B. If $\mathcal{A}$ answers k queries in $o\left(\frac{k\sqrt{n}}{\log n} + D\right)$ rounds then P is $\tilde{\Omega}(\sqrt{n})$.

Lower Bound for k-source shortest path. By removing path v_0^*-$v_{\ell-1}^*$ in the construction in Lemma 2 and slightly modifying the proof, we obtain a lower bound of $\tilde{\Omega}(\sqrt{nk} + D)$ for computing distances between k vertex pairs $d(a_i, b_i)$. Thus, we get a $\tilde{\Omega}(\sqrt{nk} + D)$ lower bound for k-source SSSP in directed unweighted graphs, proving Theorem 5. This lower bound is almost tight for $k \geq n^{1/3}$, as the k-source SSSP algorithm of [14] takes $\tilde{O}(\sqrt{nk} + D)$ rounds for k-SSSP in directed unweighted graphs. For $k < n^{1/3}$ the bound is not tight, which is not unexpected as we have a gap between the current best upper and lower bounds even for SSSP ($k = 1$), i.e., $\tilde{O}(\sqrt{n} + n^{2/5+o(1)} D^{2/5} + D)$ [6] and $\tilde{\Omega}(\sqrt{n} + D)$ [21].

3 All Pairs Second Simple Shortest Path

In the 2-APSiSP problem, we need to compute the 2-SiSP distance $d_2(x, y)$ for all pairs of vertices $x, y \in V$. We present a $\tilde{O}(n)$ round algorithm for 2-APSiSP in directed weighted graphs and show that it is almost optimal with a $\tilde{\Omega}(n)$ round lower bound even for undirected unweighted graphs with constant diameter.

2-APSiSP Algorithm. Our algorithm uses a characterization of 2-SiSP proved in [1], and we show that all 2-SiSP distances can be computed using n exclude computations followed by local computation at each vertex. Using the excluded shortest paths result from Theorem 1, we obtain our $\tilde{O}(n)$ round bound. Due to space constraints, we defer the detailed presentation of our 2-APSiSP algorithm to the full version [16].

2-APSiSP Lower Bound. We prove a lower bound of $\tilde{\Omega}(n)$ for 2-APSiSP, proving Theorem 7. A lower bound of $\tilde{\Omega}(n)$ rounds for computing directed weighted 2-SiSP for a single pair of vertices (which trivially applies to 2-APSiSP) was proven in [15]. The result in Lemma 2 of Sect. 2.3 with $k = \Theta(n)$ can also be modified to give a $\tilde{\Omega}(n)$ lower bound for 2-APSiSP. But these results only apply to directed graphs with diameter $\Omega(\log n)$. Our lower bound holds for undirected unweighted graphs with diameter as low as $O(1)$ even when APSP distances in the input graph are known, i.e., vertex $y \in V$ knows distances $d(x, y)$ for all $x \in V$.

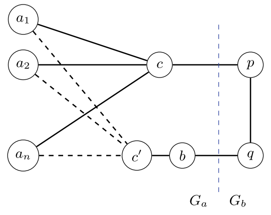

Fig. 2. 2-APSiSP lower bound.

Proof (of Theorem 7). Consider the undirected unweighted graph construction $G = (V, E)$ in Fig. 2, based on a n-bit set disjointness instance S_a, S_b. Solid edges are always present, but for each $1 \leq i \leq n$, edge (a_i, c') is present only if $S_a[i] = 1$. G has $\Theta(n)$ vertices and constant undirected diameter. Alice controls vertices a_i, c, c', b and Bob controls vertices p, q – note that Alice and Bob can construct their sections of the graph with only the bits known to them, and the crossing edges are fixed.

First, we show that set disjointness can be determined by computing 2-APSiSP in G. Note that $d(a_i, q) = 3$, either using the path through vertex c, or using the path through c' when edge (a_i, c') is present is G. If edge (a_i, c') is present, the 2-SiSP distance is the same $d_2(a_i, q) = 3$. If the edge is not present, $d_2(a_i, q) \geq 5$ using a path segment of the form a_i-c-a_j-c'. Since only 2 edges cross the cut separating vertices controlled by Alice and Bob, they can simulate a 2-APSiSP algorithm taking $R(n)$ rounds using $O(R(n) \cdot \log n)$ bits. After computing all 2-APSiSP distances, Bob can identify all bits of S_a using the distances $d_2(a_i, q)$ computed at q. Then, Bob can readily evaluate set disjointness of S_a, S_b and forward the 1-bit result to Alice in one round. So, the communication lower bound for set disjointness [3] gives an $\Omega(n/\log n)$ lower bound for 2-APSiSP. We also claim that APSP distances between vertices controlled by Alice and Bob are the same regardless of bits S_a, S_b. Only distances involving a_i could be affected by the input bits, and regardless of the presence of edge (a_i, c') we have $d(a_i, p) = 2$ and $d(a_i, q) = 3$.

We can generalize this lower bound to hold for $(3 - \epsilon)$-approximation algorithms (for any constant $\epsilon > 0$) by replacing the edges (a_i, c) and (a_i, c') with paths of length k, for a large constant k. The graph still has $\Theta(k)$ vertices, but we have $d_2(a_i, q) = k + 2$. If edge (a_i, c') is not present, then $d_2(a_i, q) \geq 3k + 2$, allowing us to determine set disjointness even if only a $(3 - \epsilon)$-approximation of $d_2(a_i, q)$ is known (by choosing $k = \Theta(1/\epsilon)$). For directed graphs, we can direct all edges from left to right, so there is no 2-SiSP path from a_i to q when the edge (a_i, c') is not present and $d_2(a_i, q) = \infty$. In this case, the lower bound holds for an arbitrarily large approximation factor.

4 Conclusion and Open Problems

In this paper, we present CONGEST upper and lower bounds for DSO and 2-APSiSP. Our bounds for 2-APSiSP are almost optimal, up to polylog factors, for even undirected unweighted graphs. There still remains a gap between upper and lower bounds for directed weighted DSO, leading to the following open problems:

- For a DSO algorithm that can answer a batch of k queries in $O(k+D)$ rounds, we present a lower bound of $\tilde{\Omega}(n)$ preprocessing rounds, and an upper bound that takes $\tilde{O}(n^{3/2})$ rounds. Can we bridge the gap between these bounds?
- For a DSO algorithm that takes $o\left(k\frac{\sqrt{n}}{\log n} + D\right)$ rounds to answer k queries, we show a lower bound of $\tilde{\Omega}(\sqrt{n})$ preprocessing rounds. We show an upper bound that takes $\tilde{O}(n)$ preprocessing rounds and answers k queries in $\tilde{O}(k\sqrt{n}+D)$. It is unlikely the lower bound can be improved for algorithms taking $O(k\sqrt{n}+D)$ rounds, as this would require an improved CONGEST SSSP lower bound, but it is an open problem whether the preprocessing can be reduced to sub-linear rounds.
- Can we achieve a tradeoff between preprocessing rounds and query response rounds, such as $\tilde{O}(n^{3/2-c})$ preprocessing rounds to answer k queries in $\tilde{O}(kn^c + D)$ rounds, for constant $0 < c < 1/2$? We achieve this for $c = 0$ and $c = 1/2$, but our techniques do not immediately extend to arbitrary c.

– Recently, we have obtained the first non-trivial results for DSO on the PRAM including a work-efficient algorithm with $\tilde{O}(mn)$ work and $\tilde{O}(n^{1/2+o(1)})$ parallel time, an algorithm that achieves $\tilde{O}(1)$ parallel time with $\tilde{O}(n^3)$ work, and an algorithm with a work-time tradeoff that can achieve sub-n^3 work and sub-$\sqrt{n}$ time [17]. Improving these results and studying DSO in other parallel models are open problems.

References

1. Agarwal, U., Ramachandran, V.: Finding k simple shortest paths and cycles. In: ISAAC. LIPIcs, vol. 64, pp. 8:1–8:12 (2016)
2. Agarwal, U., Ramachandran, V.: Fine-grained complexity for sparse graphs. In: STOC, pp. 239–252. ACM (2018)
3. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D.: An information statistics approach to data stream and communication complexity. J. Comput. Syst. Sci. **68**(4), 702–732 (2004)
4. Bernstein, A., Karger, D.R.: A nearly optimal oracle for avoiding failed vertices and edges. In: STOC, pp. 101–110. ACM (2009)
5. Bernstein, A., Nanongkai, D.: Distributed exact weighted all-pairs shortest paths in near-linear time. In: STOC, pp. 334–342. ACM (2019)
6. Cao, N., Fineman, J.T., Russell, K.: Efficient construction of directed hopsets and parallel single-source shortest paths (abstract). In: HOPC@SPAA, pp. 3–4. ACM (2023)
7. Chang, Y.J., Chen, Y., Dey, D., Mishra, G., Nguyen, H.T., Sanchez, B.: Optimal distributed replacement paths (2025). https://arxiv.org/abs/2502.15378
8. 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)
9. Ghaffari, M.: Near-optimal scheduling of distributed algorithms. In: PODC, pp. 3–12. ACM (2015)
10. Ghaffari, M., Parter, M.: Near-optimal distributed algorithms for fault-tolerant tree structures. In: SPAA, pp. 387–396. ACM (2016)
11. Ghaffari, M., Trygub, A.: A near-optimal low-energy deterministic distributed SSSP with ramifications on congestion and APSP. In: PODC, pp. 401–411. ACM (2024)
12. Leighton, F.T., Maggs, B.M., Richa, A.W.: Fast algorithms for finding O(congestion + dilation) packet routing schedules. Comb. **19**(3), 375–401 (1999)
13. Lenzen, C., Patt-Shamir, B., Peleg, D.: Distributed distance computation and routing with small messages. Distrib. Comput. **32**(2), 133–157 (2019)
14. Manoharan, V., Ramachandran, V.: Computing minimum weight cycle in the CONGEST model. In: PODC, pp. 182–193. ACM (2024)
15. Manoharan, V., Ramachandran, V.: Computing replacement paths in the CONGEST model. In: SIROCCO. LNCS, vol. 14662, pp. 420–437. Springer, Cham (2024)
16. Manoharan, V., Ramachandran, V.: Distributed distance sensitivity oracles (2024). https://arxiv.org/abs/2411.13728
17. Manoharan, V., Ramachandran, V.: Algorithms for distance sensitivity oracles on the PRAM (2025, manuscript)

18. Nanongkai, D.: Distributed approximation algorithms for weighted shortest paths. In: Symposium on Theory of Computing, STOC 2014, pp. 565–573. ACM (2014)
19. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM, USA (2000)
20. Peleg, D., Rubinovich, V.: A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. SIAM J. Comput. **30**(5), 1427–1442 (2000)
21. Sarma, A.D., et al.: Distributed verification and hardness of distributed approximation. SIAM J. Comput. **41**(5), 1235–1265 (2012)
22. Weimann, O., Yuster, R.: Replacement paths and distance sensitivity oracles via fast matrix multiplication. ACM Trans. Algorithms **9**(2), 14:1–14:13 (2013)
23. Yen, J.Y.: Finding the k shortest loopless paths in a network. Manag. Sci. **17**(11), 712–716 (1971)