Faster Replacement Paths and Distance Sensitivity Oracles

FABRIZIO GRANDONI, IDSIA, USI-SUPSI, Switzerland
VIRGINIA VASSILEVSKA WILLIAMS, Massachussetts Institute of Technology, USA

Shortest paths computation is one of the most fundamental problems in Computer Science. An important variant of the problem is when edges can fail, and one needs to compute shortest paths that avoid a (failing) edge. More formally, given a source node s, a target node t, and an edge e, a *replacement path* for the triple (s,t,e) is a shortest s-t path avoiding edge e. Replacement paths computation can be seen either as a static problem or as a data structure problem. In the static setting, a typical goal is to compute for fixed s and t, for every possible failed edge e, the length of the best replacement path around e (*replacement paths problem*). In the data structure setting, a typical goal is to design a data structure (*distance sensitivity oracle*) that, after some preprocessing, *quickly* answers queries of the form: what is the length of the replacement path for the triple (s,t,e)?

In this paper we focus on n-node directed graphs with integer edge weights in $[-M,M]$, and present improved replacement paths algorithms and distance sensitivity oracles based on fast matrix multiplication. In more detail, we obtain the following main results:

- We describe a replacement paths algorithm with runtime $\tilde{O}(Mn^{\omega})$, where $\omega < 2.373$ is the fast matrix multiplication exponent. For a comparison, the previous fastest algorithms have runtime $\tilde{O}(Mn^{1+2\omega/3})$ [Weimann,Yuster-FOCS'10] and, in the unweighted case, $\tilde{O}(n^{2.5})$ [Roditty,Zwick-ICALP'05]. Our result shows that, at least for small integer weights, the replacement paths problem in directed graphs may be easier than the related all-pairs shortest paths problem, as the current best runtime for the latter is $\tilde{O}(M^{\frac{1}{4-\omega}}n^{2+\frac{1}{4-\omega}})$: this is $\Omega(n^{2.5})$ even if $\omega = 2$. Our algorithm also implies that the k shortest simple s-t paths can be computed in $\tilde{O}(kMn^{\omega})$ time.
- We consider the *single-source* generalization of the replacement paths problem, where only the source s is fixed. We show how to solve this problem in all-pairs shortest paths time, currently $\tilde{O}(M^{\frac{1}{4-\omega}}n^{2+\frac{1}{4-\omega}})$. Our runtime reduces to $\tilde{O}(Mn^{\omega})$ for positive weights, hence matching our mentioned result for the simpler replacement paths case (that however holds also for nonpositive weights). One of the ingredients that we use is an algorithm to compute the distances from a set S of source nodes to a set T of target nodes in $\tilde{O}(Mn^{\omega} + |S|\cdot|T|\cdot(Mn)^{\frac{1}{4-\omega}})$ time. This improves on a result in [Yuster,Zwick-FOCS'05].
- We present the first distance sensitivity oracle that achieves simultaneously subcubic preprocessing time and sublinear query time. More precisely, for a given parameter $\alpha \in [0,1]$, our oracle has preprocessing time $\tilde{O}(Mn^{\omega+\frac{1}{2}} + Mn^{\omega+\alpha(4-\omega)})$ and query time $\tilde{O}(n^{1-\alpha})$. The previous best oracle for small integer weights has $\tilde{O}(Mn^{\omega+1-\alpha})$ preprocessing time and (superlinear) $\tilde{O}(n^{1+\alpha})$ query time [Weimann,Yuster-FOCS'10]. From a technical point of view, an interesting and novel aspect of our oracle is that it exploits

The first author was partially supported by the SNSF Grants APXNET 200021_159697/1 and SNSF Excellence Grant 200020B_182865/1. The second author was partially supported by NSF Grants CCF-1417238, CCF-1528078, CCF-1514339, CCF-0830797, CCF-1118083, IIS-0963478 and IIS-0904325, BSF Grant BSF:2012338 and by AFOSR MURI Grant. This work was initiated while V.V.W. was at UC Berkeley and at Stanford University. A preliminary version of this paper appeared in SODA'11 [41] and FOCS'12 [24].
Authors' addresses: FABRIZIO GRANDONI, IDSIA, USI-SUPSI, Lugano, Switzerland; VIRGINIA VASSILEVSKA WILLIAMS, Massachussetts Institute of Technology, CSAIL, Cambridge, MA, USA.

XXXX-XXXX/2019/10-ART $15.00
https://doi.org/1

as a subroutine our single-source replacement paths algorithm. We also present an oracle with the same preprocessing time as in [Weimann,Yuster-FOCS'10] and with smaller query time $\tilde{O}(n^{1-\frac{1-\alpha}{4-\omega}} + n^{2\alpha})$.

CCS Concepts: • **Theory of computation → Shortest paths**; *Data structures design and analysis.*

Additional Key Words and Phrases: replacement paths; distance sensitivity oracles; shortest paths.

ACM Reference Format:
FABRIZIO GRANDONI and VIRGINIA VASSILEVSKA WILLIAMS. 2019. Faster Replacement Paths and Distance Sensitivity Oracles. 1, 1 (October 2019), 25 pages. https://doi.org/1

1 INTRODUCTION

Shortest paths computation is one of the most fundamental problems in Computer Science. A natural generalization of the shortest paths problem for failure prone graphs is to compute shortest paths avoiding a (*failing*) edge e, so called *replacement paths* w.r.t. e. A typical motivation for replacement paths computation is to quickly recover from edge failures. Replacement paths are useful also in contexts where one may wish to satisfy other constraints beyond short length [30]. For instance, in biological sequence alignment [11] replacement paths are useful in determining which pieces of an alignment are most important. The replacement paths problem is also used in the computation of Vickrey Prices of edges that are owned by selfish agents in a network [25, 33], and in finding the k shortest simple paths between two nodes [30, 36, 37, 46].

More formally, let $G = (V, E)$ be an n-node m-edge directed graph, with integer edge weights (or lengths) $w : E \rightarrow [-M, +M]$, where M is a positive integer[1]. From now on we will always assume that the considered graph G contains no negative cycles, so that shortest paths are well-defined. If there are multiple shortest paths between two nodes in a given graph, we will implicitly consider a *canonical* shortest path obtained via some tie breaking rule. In particular, $P_{s,t}$ will denote the (canonical) shortest path from s to t in the input graph. By $dist_G(s, t)$ we denote the length of the shortest path from s to t (*distance* from s to t) in G, and we will use the shortcut $dist(s, t)$ when G is clear from the context[2]. Given two nodes s and t and an edge e, a *replacement path* $P_{s,t,e}$ for the triple (s, t, e) is the shortest path from s to t that avoids edge e, i.e., the shortest s-t path in $G \setminus \{e\}$. Observe that, if e is not an edge of $P_{s,t}$, then we can assume w.l.o.g. that $P_{s,t,e} = P_{s,t}$. Hence, w.l.o.g. we will assume from now on that e belongs to $P_{s,t}$. For the sake of simplicity, we will next focus on the computation of the lengths $D_{s,t,e} := dist_{G\setminus\{e\}}(s, t)$ of the considered replacement paths $P_{s,t,e}$. However, our approach can be adapted via standard techniques to allow for the computation of any replacement path $P_{s,t,e}$ in time linear in its number of edges. Intuitively, we can associate with each distance $D_{s,t,e}$, the predecessor of t along $P_{s,t,e}$. This way we can reconstruct $P_{s,t,e}$ proceeding backward node by node.

In the literature, there are two main high-level approaches used to compute replacement paths. In the first approach, one solves the problem *statically* by precomputing the lengths $D_{s,t,e}$ of all the (non-trivial) replacement paths $P_{s,t,e}$ and storing them in a table. Depending on the restrictions on s and t, one obtains different variants of the problem. The so called *Replacement Path* problem (RP) is obtained by fixing both s and t. In this paper we will also consider the *Single-Source Replacement Paths* problem (SSRP), where s is fixed and t is arbitrary. SSRP is a natural extension of RP, and moreover, it turns out to be useful in the design of distance sensitivity oracles.

The second approach to replacement paths is to design a *data structure* that, after a *preprocessing* step, quickly answers *queries* of the form (s, t, e) by returning $D_{s,t,e}$. Such a data structure is called a *Distance Sensitivity Oracle* (DSO). As usual, one needs to compromise between the preprocessing

[1]Throughout this paper, for integers $a \leq b > 0$, $[a, b] = \{a, a + 1, \ldots, b\}$ and $[b] = \{1, 2, \ldots, b\}$.
[2]The weight function associated with the considered graph G will always be clear from the context.

and query time[3]. In this paper we will focus only on the all-pairs variant of DSOs (where both s and t are arbitrary).

1.1 Related Work

1.1.1 Replacement Paths. RP is the best studied variant of the replacement paths problem. Recall that here both the source s and the target t are fixed. The naive way to solve RP is to remove each edge $e \in P_{s,t}$ in turn and compute the shortest path in $G \setminus \{e\}$ from scratch. This approach is however unnecessarily time-consuming as the shortest paths computations share a lot of information. RP can be solved very efficiently in undirected graphs: Malik et al. [31] gave an $\tilde{O}(m)$ time algorithm[4]. Nardelli et al. [32] used Thorup's linear time algorithm for single-source shortest paths [40] to improve the runtime to $O(m\alpha(n))$ in the word-RAM model of computation, where $\alpha(\cdot)$ is the inverse Ackermann function.

The best algorithm for the problem in *sparse* directed graphs with arbitrary edge weights is by Gotthilf and Lewenstein [23] and runs in $O(mn + n^2 \log\log n)$ time. For *dense* directed graphs, nothing much better than cubic time is known[5]. Vassilevska Williams and Williams [43] showed that RP in directed graphs is equivalent under *subcubic reductions* to the All-Pairs Shortest Paths problem (APSP), i.e. the problem of computing all the pairwise distances in a given graph. This essentially means that either both problems admit *truly subcubic* algorithms, i.e. algorithms with runtime $\tilde{O}(n^{3-\varepsilon})$ for some constant $\varepsilon > 0$, or neither of them does. It is worth pointing out that this apparent cubic time barrier is only due to the wish to compute the replacement distances *exactly*. In contrast, Bernstein [3] described an algorithm for RP in directed graphs with positive weights that for any constant $\varepsilon > 0$, computes $(1+\varepsilon)$-approximate replacement paths in $\tilde{O}(\frac{1}{\varepsilon}m)$ time. For weighted planar digraphs, the runtime can be reduced to $\tilde{O}(n)$ as shown by Emek, Peleg and Roditty [18].

For *unweighted* directed graphs, Roditty and Zwick [37] gave a randomized combinatorial algorithm that computes replacement paths in $\tilde{O}(m\sqrt{n})$ time[6]. Weimann and Yuster [44] applied fast matrix multiplication techniques to the problem. Their randomized algorithm runs in $\tilde{O}(Mn^{1+2\omega/3})$ time, where $\omega < 2.373$ [21, 42] is fast square matrix multiplication exponent defined as the smallest constant such that two $n \times n$ matrices can be multiplied using $\tilde{O}(n^\omega)$ elementary operations[7]. Using rectangular matrix multiplication algorithms [14, 20, 27], the running time can be slightly improved to $O(Mn^{2.584})$. Observe that, if $\omega = 2$, Weimann and Yuster's running time would be $O(Mn^{2.334})$. This would improve on Roditty and Zwick's $O(n^{2.5})$ running time (assuming $\omega = 2$) for dense unweighted graphs.

Somehow surprisingly, SSRP (where only the source s is fixed) has not received much attention in the literature. The only reference to our knowledge is a paper by Hershberger et al. [26] that

[3]Another important aspect is the space complexity: this is not the main focus of this paper, and we will only have a brief discussion of it.

[4]For notational convenience, throughout this paper we use a modified $\tilde{O}$ notation, which suppresses sub-polynomial (rather than just poly-logarithmic) factors in Mn. However, the reader should be aware that in several cases the hidden factor is only poly-logarithmic.

[5]Subpolynomial improvements are known. The best of these is achieved by combining the reduction from Replacement Paths to APSP presented in this paper with the current fastest algorithm for APSP by Williams [45] running in $n^3/2^{\Theta(\sqrt{\log n})}$ time, leading to the same running time for RP.

[6]The Roditty and Zwick algorithm can be extended to support integer edge weights in $[1, M]$ in time $\tilde{O}(m\sqrt{Mn})$: The algorithm processes detours shorter than L in $\tilde{O}(mL)$ time, and this works even in the weighted case. It processes detours longer than L by sampling $O(n \log n/L)$ vertices in order to hit every long detour. In the weighted case we need to sample $O(nM \log n/L)$ vertices instead. To obtain the new running time, one sets $L = \sqrt{nM}$.

[7]The value ω is defined for the arithmetic circuit model where the elementary operations are multiplication and addition of elements from an underlying field such as the complex numbers.

refers to the problem as *edge-replacement shortest paths trees* and shows that in the path-comparison model of computation of Karger et al. [29], SSRP on directed graphs with arbitrary edge weights requires $\Omega(mn)$ comparisons. The aforementioned reduction from APSP to RP by [43] suggests that there is little hope for a truly subcubic algorithm for SSRP. SSRP is a natural problem, and moreover, it is a basic primitive for designing DSOs, as we will describe in this paper.

The RP problem is closely related to the problem of finding the second shortest path between two nodes, and in general to the k *shortest paths* problem. For directed graphs and nonnegative edge weights, Eppstein [19] gave an algorithm that returns the k shortest paths from s to t in time $O(m + n \log n + k)$. The paths that Eppstein's algorithm returns, however, may not be simple. When the k shortest paths are required to be simple, the fastest known algorithms for the problem use algorithms for replacement paths. In more detail, Roditty and Zwick [37] showed that the k simple shortest paths problem can be reduced to $O(k)$ computations of the second shortest simple path, and hence to the solution of $O(k)$ instances of RP.

1.1.2 Distance Sensitivity Oracles. DSOs are very well-studied in the literature. For arbitrary *nonnegative* edge weights, there are two trivial approaches. The first does no precomputation and each query (s, t, e) is answered by computing the shortest path between s and t in $G \setminus \{e\}$ explicitly in $O(m + n \log n)$ time using Dijkstra's algorithm. The second approach takes $\tilde{O}(mn^2)$ preprocessing time to compute for every source node s and for every edge e in the shortest paths tree rooted at s, the new shortest paths tree from s in $G \setminus \{e\}$. The queries are then answered in $O(1)$ time by looking up the stored solutions. Similar DSOs can be obtained for graphs with possibly negative weights but no negative cycles by adding an extra preprocessing step to replace all negative weights by nonnegative ones, as in [28]. This preprocessing step either takes $\tilde{O}(mn)$ time using the Bellman-Ford algorithm, or $O(m\sqrt{n} \log M)$ time using Goldberg's scaling algorithm [22].

The preprocessing time for DSOs with arbitrary edge weights was improved to $\tilde{O}(mn^{\frac{3}{2}})$ by Demetrescu et al. [15], while keeping the query time constant. Bernstein and Karger further improved the preprocessing time to $\tilde{O}(\sqrt{mn^2})$ [4] and finally to $\tilde{O}(mn)$ [5]. The latter preprocessing time matches, up to poly-logarithmic factors, the best known runtime for APSP in the same setting, and seems therefore very hard to beat.

One can do better, at least in terms of preprocessing time, in the case of integer weights of small absolute value. Weimann and Yuster [44] presented a DSO with preprocessing time $\tilde{O}(Mn^{\omega+1-\alpha})$ and query time $\tilde{O}(n^{1+\alpha})$ for any given parameter $\alpha \in [0, 1]$. In particular, they showed that the problem can be solved with both subcubic preprocessing time and subquadratic query time[8]. An obvious open problem is whether one can achieve subcubic preprocessing time with linear (or even sublinear) query time. We answer this question affirmatively.

In this case one can also do better in terms of running time if one is allowed for a multiplicative or additive error in the reported replacement path lengths (in some cases for general weights and/or for multiple faults) [2, 12, 17]. In some cases, this is a direct consequence of *fault-tolerant spanner* results [7–10, 13, 16, 34, 35]. We remark that the requirement of computing the replacement path lengths exactly makes the problem substantially harder.

1.2 Our Results and Techniques

In this paper we present faster replacement paths algorithms and distance sensitivity oracles in the case of dense graphs with small integer weights. All our results exploit fast matrix multiplication procedures. In more detail, we achieve the following main results.

[8] [44] also considers the case of $f = O(1)$ (simultaneous) failures: part of our results can be extended in that direction, but this is out of the scope of this paper.

1.2.1 Replacement Paths. Improving on [37, 44], we present a faster algorithm for the classical RP problem.

Theorem 1.1. *There is a randomized algorithm that solves RP in n-node directed graphs with integer weights in $[-M, M]$ in $\tilde{O}(Mn^{\omega})$ time, with failure probability[9] polynomially small in n.*

Theorem 1.1 improves on Roditty and Zwick's $O(n^{2.5})$ runtime for dense unweighted graphs. It also improves the range of M for which there is a subcubic algorithm for the problem: in the case of Weimann and Yuster's algorithm it is roughly $M \leq n^{0.416}$, while for our algorithm it is roughly $M \leq n^{0.627}$. Our result also shows that, at least for small integer weights, the replacement paths problem in directed graphs might actually be easier than APSP in directed graphs. In more detail, Zwick's algorithm [49] solves APSP in $\tilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$ time[10]. Note that this runtime is $\Omega(n^{2.5})$ even in unweighted directed graphs and assuming $\omega = 2$. Furthermore, improving on $O(n^{\omega})$ for replacement paths in directed unweighted graphs would likely require radically new techniques, as the problem is closely related to Boolean matrix multiplication.

As a consequence of the aforementioned reduction from k shortest simple paths to RP in [37], we obtain the following corollary.

Corollary 1.2. *There is a randomized algorithm that solves k shortest simple paths in a directed n-node graph with integer edge weights in $[-M, M]$ in $\tilde{O}(kMn^{\omega})$ time, with failure probability polynomially small in n.*

Our result is based on two main ingredients. On the one hand, we exploit a simple (deterministic) reduction from RP to APSP. On the other hand, we design a divide-and-conquer randomized recursive strategy.

In more detail, as in previous work we rely on the notion of *detour*. Let $P = P_{s,t} = \{s = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k = t\}$ be the considered shortest path from s to t in G. We next assume that P is given, as well as all the distances between pairs of nodes along P. This can be easily computed in $\tilde{O}(n^2)$ extra time. For $k > j$, a *detour* $\Delta(v_j, v_k)$ between v_j and v_k is a shortest path from v_j to v_k that does not contain any other node of P. This detour is said to *circumvent* every edge between v_j and v_k in P. It is well known (see e.g. [3], Lemma 2.1) that for any edge $e_i := (v_i, v_{i+1}) \in E(P)$, the shortest path between s and t in $G \setminus \{e_i\}$ is exactly the minimum out of all paths of the form

$$s \rightarrow v_2 \rightarrow \ldots \rightarrow v_j \odot \Delta(v_j, v_k) \odot v_k \rightarrow \ldots \rightarrow t,$$

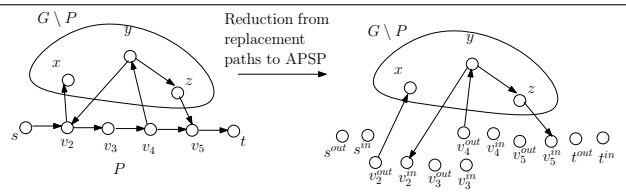
where $j \leq i$, $i + 1 \leq k$, and $\odot$ denotes concatenation.

Suppose that we are given for every $v_j, v_k \in P$ $(j < k)$, the length of the detour $\Delta(v_j, v_k)$. We will describe how to compute the lengths of all the replacement paths in only $O(n^2 \log n)$ extra time. First, we compute in $O(n^2)$ time for every detour $\Delta(v_j, v_k)$, the length $\ell(v_j, v_k)$ of the path $s \rightarrow \ldots \rightarrow v_j \odot \Delta(v_j, v_k) \odot v_k \rightarrow \ldots \rightarrow t$, as follows. As we are given P, we can in linear time compute the length of each subpath $s \rightarrow \ldots \rightarrow v_j$ for all j and the length of each subpath $v_k \rightarrow \ldots \rightarrow t$ for all k. Then $\ell(v_j, v_k)$ can be computed in constant time by adding the lengths of the two paths and that of the detour. After this, we sort the $O(n^2)$ triples $(v_j, v_k, \ell(v_j, v_k))$ in nondecreasing order according to $\ell(\cdot)$, in $O(n^2 \log n)$ time. Furthermore, we store all the pairs (v_i, i) with $v_i \in P$ in a successor search data structure T (e.g. any balanced binary search tree), with search key i. Intuitively, each such pair (v_i, i) in T corresponds to an edge e_i for which we did not compute the replacement path length yet. We then scan the triples $(v_j, v_k, \ell(v_j, v_k))$ according to

[9] All the algorithms considered in this paper return lengths that are never smaller than the correct ones: the failure probability refers to the event that a strictly larger length is returned.

[10] The runtime of Zwick's algorithm can be slightly improved to $O(M^{0.681} n^{2.532})$ by using fast rectangular matrix multiplication [20].

Figure 1 An example of the reduction from RP to APSP in the case of an unweighted graph.



the aforementioned sorted order. For any such triple, we find all the pairs (v_i, i) still in T with $j \le i \le k-1$ (in logarithmic time per pair). For any such pair (v_i, i), we record that the shortest replacement path length[11] for e_i is $\ell(v_j, v_k)$, and then delete (v_i, i) from T. Intuitively, all triples $(v_j, v_k, \ell(v_j, v_k))$ with $j \le i \le k-1$ induce a candidate replacement path of length $\ell(v_j, v_k)$ for e_i. By construction, among these options, we consider only the shortest one.

Given the above construction, a reduction to APSP can be easily achieved as follows[12]. For every node v_i of P, create two copies v_i^{in} and v_i^{out}. Create a new graph G' by taking $(G \setminus P) \cup \{v_i^{in}, v_i^{out}\}_{i \in [k]}$. For every edge (v_i, u) for $u \in G \setminus P$, add an edge (v_i^{out}, u) of the same weight in G'. Similarly, for every edge (u, v_i) for $u \in G \setminus P$, add an edge (u, v_i^{in}) of the same weight in G'. G' is essentially G with the edges of P removed, except that each node of P is split into two.[13] See Figure 1 for an example conversion from G to G'. Now solve APSP in G'. The shortest path between v_i^{out} and v_j^{in} is exactly the optimal detour $\Delta(v_i, v_j)$ in G. Thus with one call to an APSP algorithm, and $O(n^2 \log n)$ extra time we obtain an algorithm for RP. Applying Zwick's [49] algorithm for APSP, we can solve the problem in $\tilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$ (deterministic) time.

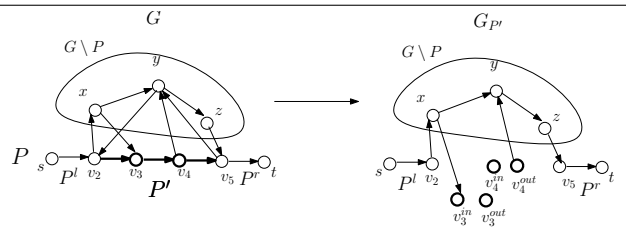
We are able to achieve a runtime strictly better than Zwick's APSP algorithm via a bucketting argument. Let us partion P into q subpaths $P_1, \ldots, P_q$ of size roughly n/q, for a carefully chosen parameter q. Let us focus on the edges of one such subpath P' between nodes v_x and v_y. We construct an auxiliary graph $G_{P'}$ as follows. Let P^l and P^r be the subpaths of P to the left and right of P' respectively. Take G and remove all incoming edges to nodes on P^l except those in P and all outgoing edges from nodes on P^r except those in P. This will ensure that any path that we compute exiting P^l does not reenter it and any path entering P^r does not reexit it. Now remove all edges in P' and split each node v in $W(P') := V(P') - \{v_x, v_y\}$ into two as before: a copy v^{in} with all the remaining incoming edges and a copy v^{out} with all the remaining outgoing edges. An example of the construction of $G_{P'}$ is given in Figure 2. Let us compute the shortest path lengths from s and to t in $G_{P'}$ in $\tilde{O}(n^2)$ time. The shortest replacement path from s to t avoiding all edges in P' is simply the shortest s-t path in $G_{P'}$. Consider now the shortest replacement path that leaves P not later than v_x and reenters P at some node $v_i \in W(P_i)$. This path is obtained by appending to the shortest s-v_i^{in} path in $G_{P'}$ the subpath of P between v_i and t. Symmetrically, one can compute the shortest replacement path that leaves at some node $v_i \in W(P')$ and reenters P not earlier than v_y. The only remaining replacement paths for edges in P' have a detour with both endpoints in $W(P')$. They can be derived by computing the shortest paths distances between copies of nodes in $W(P')$ in $G_{P'}$. As we will discuss later, we are able to perform the latter computation in time $\tilde{O}(Mn^\omega + M^{\frac{1}{4-\omega}} n^{\frac{3}{4-\omega}} |W(P')|^{2-\frac{2}{4-\omega}})$. Therefore the total computation time is $\tilde{O}(qMn^\omega + qM^{\frac{1}{4-\omega}} n^{\frac{3}{4-\omega}} (\frac{n}{q})^{2-\frac{2}{4-\omega}})$.

[11] The actual path can also be stored, as usual, with a matrix of successors.

[12] The possibility of such a reduction was mentioned in [6]; here we make it explicit and show that it can be used to further improve the known runtime bounds.

[13] Splitting the nodes in two is not strictly necessary. Our algorithms would work even without the splitting. However, the analysis is simpler with the splitting since this way all computed detours are disjoint from P.

Figure 2 An example of the transformation of a graph G into $G_{P'}$, given P and v_2-v_5 subpath P'. Notice that edges (y, v_2) and (v_5, y) are removed, and that all nodes in $W(P') = V(P') - \{v_2, v_5\}$ get split into two. The relevant paths in $G_{P'}$ corresponding to detour paths circumventing edges in P are $s \to v_2 \to x \to v_3^{in}$, $s \to v_2 \to x \to y \to z \to v_5 \to t$ and $v_4^{out} \to y \to z \to v_5 \to t$.



Choosing $q = \Theta(\sqrt{(Mn)^{\frac{1}{4-\omega}}/(Mn^{\omega-2})})$, so that $n/q = \Theta(\sqrt{Mn^{\omega}/(Mn)^{\frac{1}{4-\omega}}})$, the overall runtime of the algorithm is $\tilde{O}(M^{\frac{1}{2}(1+\frac{1}{4-\omega})}n^{1+\frac{1}{2}(\omega+\frac{1}{4-\omega})})$. This is strictly faster than Zwick's APSP algorithm.

In order to achieve the claimed $\tilde{O}(Mn^{\omega})$ runtime, we use recursion in combination with a randomized *contraction step*. The idea is to partition P into Z subpaths as in the bucketting algorithm. However, here Z is sub-polynomial (rather than polynomial). For each subpath P_i, we construct a contracted version $G(P_i)$ of the input graph, with slightly fewer nodes and slightly larger edge weights. Graph $G(P_i)$ preserves the replacement paths lengths w.r.t. the edges of P_i, and it can be computed efficiently.

1.2.2 Single-Source Replacement Paths. We present the first subcubic algorithms for SSRP in case of small integer weights. Recall that Hershberger et al. [26] gave a cubic lower bound for the problem in the path-comparison model of Karger et al. [29]. We avoid this lower bound due to our use of fast matrix multiplication which falls outside the path-comparison model.

Theorem 1.3. *There is a randomized algorithm that solves SSRP in n-node directed graphs with integer weights in $[-M, M]$ in time $\tilde{O}(M^{\frac{1}{4-\omega}}n^{2+\frac{1}{4-\omega}})$. For positive weights the runtime can be reduced to $\tilde{O}(Mn^{\omega})$. The failure probability is polynomially small in n.*

We remark that the runtime of our SSRP algorithm for integer weights in $[-M, M]$ matches the runtime of Zwick's APSP algorithm [49]. We also remark that the runtime of our algorithm for positive weights matches our own improved result for the simpler RP problem. We suspect that the case in which the weights can be negative might be intrinsically harder, as the problem seems to be more tightly related to APSP. Showing that this is the case, or obtaining an $\tilde{O}(Mn^{\omega})$ algorithm for possibly negative weights as well, is an interesting open problem.

We give some intuition about our approach in the following. Let T_s be the shortest paths tree from source s. $P_{v,u}$ denotes the path from v to u in T_s, and $dist(v, u)$ its length. For a pair $(t, e) \in V \times E$, if e does not lie along $P_{s,t}$, then $P_{s,t,e} = P_{s,t}$. We call the remaining pairs (t, e) *relevant*, and focus on them. The first step in our algorithm is a partition of T_s into a small (subpolynomial) number of subtrees T'. Using balanced tree separators, we can guarantee that each T' contains roughly the same number of nodes (modulo constants). Let P' be the path from s to the root of T'. For any relevant pair (t, e) there must exist some subtree T' such that $t \in V(T')$ and either (a) $e \in E(T')$ or (b) $e \in E(P')$. This way we identify a collection of subproblems, where each subproblem is of the following two forms:

- In a *subtree problem*, we are given a subtree T' of T and we want to compute replacement paths $P_{s,t,e}$ where both t and e belong to T' (handling (a) above).

- In a *subpath problem* we are given a subpath P' of T from the source s to a node t', and a subtree T' of T rooted at t', and we want to compute replacement paths $P_{s,t,e}$ with t in T' and e in P' (handling (b) above).

We solve each subtree problem T' recursively, after a preliminary *compression* step where we replace the nodes outside T' with a subpolynomially smaller random subset B of them, adding auxilliary edges (with subpolynomially larger weights) representing shortest paths between the sampled nodes.

Handling subpath problems (P', T') is the crux of our approach. The portion of $P_{s,t,e}$ not in $P_{s,t}$ is called a *detour* and (w.l.o.g.) is a path that starts at some node v of $P_{s,t}$ (before edge e) and ends at some other node u of $P_{s,t}$ (after edge e). Note that possibly $v = s$ and/or $u = t$. For a given subpath problem (P', T'), we distinguish two types of replacement paths $P_{s,t,e}$ depending on their detour:

- In *jumping* paths, detours have both endpoints in P';
- In *departing* paths, detours have only the starting node in P'.

We can reduce the computation of jumping paths to an instance of the RP problem which we solve in $\tilde{O}(Mn^{\omega})$ time with our own RP algorithm.

The computation of departing paths essentially reduces to the computation of their detours. For positive weights, we are able to compute such detours in $\tilde{O}(Mn^{\omega})$ time. We adapt an idea of Roditty and Zwick [37] used in their unweighted RP algorithm. Roughly speaking, consider the detour $\tilde{P}_{v,u}$ of a departing path $P_{s,t,e}$, going from some $v \in V(P')$ to some $u \in V(T') - \{t'\}$. Suppose that $\tilde{P}_{v,u}$ has X nodes and hence length at most MX. Consequently also the length of the shortest path $P_{v,u}$ from v to u is at most MX, which implies that $P_{v,u}$ contains at most MX nodes (here we exploit the positiveness of the weights). This forces v to be one of the final MX nodes of P' (since $u \notin V(P')$). We exploit the above observation as follows. Let L be a proper integer threshold. We compute all the distances in $G - E(P')$ from the final ML nodes of P' to $V(T')$: this way we obtain the detours with $X \leq L$. Then we sample a random set B of $\tilde{O}(n/L)$ nodes, so that w.h.p. B hits all the detours with $X \geq L$. We compute the shortest paths from $V(P')$ to B and from B to $V(T')$, and then derive the desired detour lengths by going through all the triples $(v, b, u) \in V(P') \times B \times V(T')$.

Consider the computation of shortest paths in the two stages of the algorithm. In both cases we have to solve an instance of the following *S-T shortest paths* problem (STSP): given a directed edge-weighted graph $G = (V, E)$, and two subsets of nodes $S, T \subseteq V$, compute all the distances between pairs $(s, t) \in S \times T$. The best know algorithm for STSP (for M small enough) is given in [47] and has runtime $\tilde{O}(Mn^{\omega} + M^{\frac{1}{4-\omega}} n^{\frac{3}{4-\omega}} (|S|\,|T|)^{1-\frac{1}{4-\omega}})$. We improve this to:

Theorem 1.4. *There is a randomized algorithm that solves STSP in n-node directed graphs with integer weights in* $[-M, M]$ *in time* $\tilde{O}(Mn^{\omega} + |S| \cdot |T| \cdot (Mn)^{\frac{1}{4-\omega}})$, *with failure probability polynomially small in n.*

Incidentally, Yuster and Zwick mention that with their distance oracle one can compute shortest paths trees from $\tilde{\Theta}(Mn^{w-2})$ sources in $\tilde{O}(Mn^{w})$ time. We can do the same from $\tilde{\Theta}(M^{1-\frac{1}{4-\omega}} n^{\omega-1-\frac{1}{4-\omega}})$ sources, which is $\tilde{\Theta}(\sqrt{n})$ even for $M = O(1)$ and $\omega = 2$, whereas the number of sources Yuster and Zwick can handle is only $\tilde{\Theta}(1)$ in that case.

Using our STSP algorithm and choosing L properly, we are able to solve a subpath problem in time $\tilde{O}(Mn^{\omega} + M^{\frac{1}{2}+\frac{1}{2(4-\omega)}} n^{2+\frac{1}{2(4-\omega)}})$. This is $\tilde{O}(Mn^{\omega})$ for ω big enough (in particular it holds for the current best upper bound, namely 2.373). In order to obtain a runtime of $\tilde{O}(Mn^{\omega})$ for any value of ω, we use a scaling trick. We consider a logarithmic subset of intervals $[X, 2X) = [X, 2X - 1]$, $X \geq L$, and search for the detours with a number of nodes in the interval by considering the detours

that start in the last $2MX$ nodes of P' and pass through a sample of $\tilde{O}(n/X)$ nodes. This way, going through the triples (v, b, u) costs only $\tilde{O}(Mn^2)$ rather than $\tilde{O}(n^3/L)$.

For arbitrary weights the idea of considering the final MX nodes of P' does not work: here a very low weight path might contain many nodes due to negative (or even zero) edge weights. In this case we simply solve APSP in $G - E(P')$ with Zwick's algorithm (using our STSP algorithm does not help since possibly $|V(P')| = \Omega(n) = |V(T')|$). This solves SSRP for integer weights in $[-M, M]$ in the claimed $\tilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$ time.

1.2.3 Distance Sensitivity Oracles. In this paper we present the first distance sensitivity oracle that achieves simultaneously subcubic preprocessing time and sublinear query time.

Theorem 1.5. *For any integer $1 \leq S \leq n$, there is a randomized distance sensitivity oracle for n-node directed graphs with integer weights in $[-M, M]$, with preprocessing time $\tilde{O}(Mn^\omega \cdot (S^{4-\omega} + \sqrt{n}))$, query time $\tilde{O}(\frac{n}{S})$, and failure probability polynomially small in n.*

In particular, by choosing $S = \Theta(n^{\frac{1}{2(4-\omega)}})$, one obtains subcubic preprocessing time $\tilde{O}(Mn^{\omega+\frac{1}{2}}) < O(Mn^{2.873})$ and sublinear query time $\tilde{O}\left(n^{1-\frac{1}{2(4-\omega)}}\right) < O(n^{0.693})$. Recall that the oracle by [44] has preprocessing time $\tilde{O}(Mn^{\omega+1-\alpha})$ and query time $\tilde{O}(n^{1+\alpha})$, for any given parameter $\alpha \in [0, 1]$. Our oracle is always better than that in terms of query time, and for $\alpha < \frac{1}{2}$ it improves also on the preprocessing time.

The DSO in [44] distinguishes between *hop-short* replacement paths, which contain at most $L = \Theta(n^{1-\alpha})$ nodes, and the remaining *hop-long* paths. Hop-short path lengths are computed in $\tilde{O}(n)$ time (at query time) by considering $\tilde{O}(L)$ (properly chosen) random subgraphs, and precomputing for each such graph the distance oracle in [47] in $\tilde{O}(Mn^\omega)$ time. For hop-long replacement paths $P_{s,t,e}$, the algorithm exploits a more involved procedure, based on the computation of an s-t shortest path in a proper auxiliary graph, whose construction (at query time) takes superlinear time $\tilde{O}(n^2/L)$.

We are able to reduce the query time for hop-short paths by a careful use of known techniques. In order to address hop-long paths, we exploit a completely different approach. Let B be a random sample of $\tilde{O}(n/L)$ nodes, so that w.h.p. every hop-long replacement path $P_{s,t,e}$ contains some node $b \in B$. Observe that the portion of $P_{s,t,e}$ from s to b must be the replacement path for the triple (s, b, e). Similarly for the triple (b, t, e). Suppose then that, for every $b \in B$, we precomputed the quantities $D_{s,b,e}$ and $D_{b,t,e}$. Then we can trivially answer the query (s, t, e) by computing $D_{s,t,e} = \min_{b \in B}\{D_{s,b,e} + D_{b,t,e}\}$. This takes $\tilde{O}(|B|) = \tilde{O}(\frac{n}{L})$ time, which is sublinear. Note that the computation of $D_{v,b,e}$ is equivalent to the computation of $D_{b,v,e}$ after reversing all edge directions.

From the above discussion, we can reduce the problem of designing an improved DSO to the problem of efficiently solving SSRP for all source nodes $s \in B$. In the case of positive weights, the claimed result can be obtained directly by exploiting our $\tilde{O}(Mn^\omega)$ time SSRP algorithm. We can achieve the same performance of the DSO for positive weights also in the case of arbitrary weights (despite the fact that we are not able to solve SSRP equally fast in the two cases) by means of a variant of our SSRP algorithm. Recall that we need to compute hop-long replacement paths containing at least L nodes, for a proper integer threshold L. Also in this case we exploit a scaling trick: for a logarithmically large set of intervals $[X, 2X)$, $X \geq L$, we address the problem of computing replacement paths with a number of nodes in the considered interval. To do this, we sample $\tilde{O}(n/X)$ random nodes B, and solve SSRP on each $b \in B$, but only considering replacement paths on at most $2X$ nodes. The last assumption allows us to reduce the runtime of the SSRP algorithm, since in each subpath problem (P', T') we need to consider only detours of departing paths that start from the *last* $2X$ nodes of P' (otherwise the corresponding replacement paths would have $> 2X$ nodes). The runtime of the modified SSRP algorithm turns out to be $\tilde{O}(Mn^\omega + XM^{\frac{1}{4-\omega}} n^{1+\frac{1}{4-\omega}})$. For increasing X,

each execution of the modified SSRP algorithm becomes more expensive, but this is compensated by the smaller number of executions (i.e., $\tilde{O}(n/X)$).

Incidentally, we are also able to obtain a variant of the DSO in [44] with the same preprocessing time, but with an improved query time.

THEOREM 1.6. *For any integer $1 \leq L \leq n$, there is a randomized distance sensitivity oracle for n-node directed graphs with integer weights in $[-M, M]$, with preprocessing time $\tilde{O}(LMn^{\omega})$, query time $\tilde{O}(n/L^{\frac{1}{4-\omega}} + (n/L)^2)$, and failure probability polynomially small in n.*

Choosing $L = \Theta(n^{1-\alpha})$, this gives $\tilde{O}(Mn^{\omega+1-\alpha}) \leq O(Mn^{3.373-\alpha})$ preprocessing time and $\tilde{O}(n^{1-\frac{1-\alpha}{4-\omega}} + n^{2\alpha}) \leq \tilde{O}(n^{0.386+0.615\alpha} + n^{2\alpha})$ query time. While this DSO does not involve drastically new techniques on top of the techniques needed for Theorem 1.5, we present it for the sake of completeness since it implies a strict improvement on [44] also for $\alpha > \frac{1}{2}$.

While we state our results as randomized algorithms, we note that using Zwick's bridging set techniques [49] and the modification in Section 8 of [47], our results can be derandomized with no significant loss in the running time.

1.3 Organization

The rest of this paper is organized as follows. In Section 2 we introduce some preliminary definitions and results that will be needed in the rest of the paper. In Section 3 we present our S-T shortest paths algorithm. In Sections 4 and 5 we describe our algorithms for RP and SSRP, respectively. Finally, in Section 6 we present our distance sensitivity oracles.

2 PRELIMINARIES

We use standard graph notation. When the base of a logarithm is not specified, we assume it to be 2. Throughout this paper *with high probability* (w.h.p.) means with probability at least $1 - n^{-Q}$ for some constant $Q > 0$. By adapting the constants in our algorithm, we can make Q arbitrarily large. As mentioned earlier, we use a modified $\tilde{O}$ notation which supresses $n^{o(1)}$ factors.[14]

2.1 Matrix Multiplication and Shortest Paths

Matrix multiplication is a common tool to solve shortest path problems in the presence of small integer weights. One of the key ingredients to that aim is the notion of distance product of two matrices. The distance product $A \star B$ of an $a \times b$ matrix A by a $b \times c$ matrix B is the $a \times c$ matrix C such that $C_{ij} = \min_{k=1,\ldots,b}\{A_{ik} + B_{kj}\}$. Alon, Galil and Margalit [1], following Yuval [48], show the following result:

LEMMA 2.1. *The distance product of an $a \times b$ matrix by a $b \times c$ matrix, where each entry is in $[-M, M] \cup \{+\infty\}$, can be computed in time $\tilde{O}(\min\{abc, M \cdot \frac{abc}{(\min\{a,b,c\})^{3-\omega}}\})$. In particular, for $a = b = c = n$, this runtime is $\tilde{O}(\min\{n^3, Mn^{\omega}\})$.*

Based on the above result, and exploiting a clever random sampling approach, Zwick [49] obtained the currently fastest APSP algorithm for directed graphs with small integer weights (more details about this algorithm are given later).

THEOREM 2.2. *[49] Given a directed n-node graph G with integer weights in $[-M, M]$, one can solve APSP in G in time $\tilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$.*

[14] This notation is particularly useful for algorithms based on matrix multiplication, as the matrix multiplication exponent ω is defined as an infimum, and the fastest $n \times n$ matrix multiplication algorithm really runs in $n^{\omega+o(1)}$ time anyway.

This improves and generalizes earlier work by Alon et al. [1]. The runtime of Zwick's algorithm can be slightly improved to $O(M^{0.681}n^{2.532})$ by using fast rectangular matrix multiplication [20, 27]. Similar improvements can be achieved for most of our results, but we omit the straightforward technical details here. We remark that in undirected graphs APSP can be solved faster, namely in $\tilde{O}(Mn^{\omega})$ time [38, 39].

Sometimes we will need to compute only a restricted subset of shortest paths. To this aim, the following result in [47] turns out to be useful.

Lemma 2.3. *[47] Given a directed n-node graph G with integer weights in $[-M, M]$, one can compute in $\tilde{O}(Mn^{\omega})$ time an $n \times n$ matrix D, so that the (i, j) entry of the distance product $D \star D$ is the distance between nodes i and j in G. Furthermore, by the properties of D, the length of a shortest s-t path containing at least L nodes can be computed in $\tilde{O}(n/L)$ extra time.*

The final claim of the previous lemma is only implicit in [47], but it is crucial for the design of our improved DSOs. We remark that the above result can also be interpreted as a *Distance Oracle* that, after a $\tilde{O}(Mn^{\omega})$ time preprocessing step, answers queries of the form (s, t, L) by returning the length of the shortest s-t path containing at least L nodes in $\tilde{O}(n/L)$ time (hence, as a special case, the shortest path length can be derived in $\tilde{O}(n)$ time). The techniques from the above lemma can also be used to solve the *Single-Source Shortest Paths* problem (SSSP), asking to compute the distances from a given source node s to all other vertices (and an associated shortest path tree T_s).

Corollary 2.4. *[47] Given a directed n-node graph G with integer weights in $[-M, M]$, one can solve SSSP in $\tilde{O}(Mn^{\omega})$ time.*

2.2 Hitting Hop-Long Paths

In our algorithms we sometimes use a random sampling approach to deal with *long* paths in a faster way. One key ingredient in our analysis will be the following lemma, which is inspired by [44]. We remark that, while the upper bound L in the following lemma is used similarly in the DSO by [44], the lower bound $L/8$ is crucial to analyze our improved DSO from Theorem 1.6.

Lemma 2.5. *Given a set $\mathcal{P}$ of paths and two parameters $1 \leq L \leq n$ and $N > 0$, sample $4Nn/L$ nodes B uniformly at random[15]. With probability at least $1 - |\mathcal{P}|ne^{-N}$ for each $P \in \mathcal{P}$ there exists $B(P) \subseteq B$ which partitions P into subpaths of at least $\min\{|V(P)|, L/8\}$ and at most L nodes each.*

Proof. We show that the probability that the considered event does not hold for a fixed path P from s to t is at most ne^{-N}: the full claim follows from the union bound. If P contains at most L nodes the claim is trivially true by choosing $B(P) = \emptyset$. Otherwise, let us iteratively remove the first $L/2$ nodes of P until the remaining path contains less than $L/2$ nodes. This partitions P into at most $\frac{2n}{L}$ intervals containing $L/2$ nodes and a last interval which contains between 0 and $L/2 - 1$ nodes. Let us consider the central subinterval of $L/4$ nodes of each interval, except possibly the last one.

Suppose that each central subinterval contains some node from B. In that case we let $B(P)$ consist of one arbitrary node of B per subinterval. It is not hard to see that two consecutive nodes in $B(P) \cup \{s, t\}$ in the order induced by P are at hop-distance at least $L/8$ and at most L. From the union bound, the probability that there exists one subinterval not hit by B is at most $\frac{2n}{L}(1 - \frac{L}{4n})^{\frac{4Nn}{L}} \leq ne^{-N}$. □

Corollary 2.6. *Let N and n' be such that $N \geq n \geq n' \geq 1$, and let $C > 0$ be any constant. Let B be a random sample of $(C + 3)n' \ln N$ nodes of V. With probability at least $1 - 1/N^C$, the nodes of B touch all detours on at least n/n' edges, so that the nodes of B partition every detour into subpaths on at most $2n/n'$ edges each.*

[15] We can sample B with replacement.

Proof. It is sufficient to apply Lemma 2.5 with $L = 2n/n'$ and observe that the number of the considered detours is at most n^2. □

2.3 Distance Sensitivity Oracles

The following lemma is at the heart of the DSO in [44], and it will be crucial for us as well.

Lemma 2.7. *[44] For an integer $1 \leq L \leq n$ and a constant $C > 0$, sample $s = L \cdot C \log n$ graphs $\{G_1, \ldots, G_s\}$, where each G_i is obtained from G by independently removing each edge with probability $1/L$. For C large enough, the following two claims hold w.h.p.: (a) For any edge $e \in G$, there are $\Theta(\log n)$ graphs G_i not containing e. (b) For any replacement path $P_{s,t,e}$ on at most L nodes, there is at least one G_i that does not contain e but contains $P_{s,t,e}$.*

3 A FASTER STSP ALGORITHM

Recall that in the *S-T shortest paths* problem (STSP) we are given a directed edge-weighted graph $G = (V, E)$, and two subsets of nodes $S, T \subseteq V$. Our goal is to compute all the distances between pairs $(s, t) \in S \times T$.

Our STSP algorithm builds upon Zwick's APSP algorithm [49], by combining a new idea with an approach from [47]. Let us start by sketching how Zwick's algorithm works. For a matrix D and $V', V'' \subseteq V$, let $D[V', V'']$ denote the matrix obtained by considering only the rows and columns indexed by V' and V'', respectively. The algorithm consists of a sequence of iterations $i = 1, \ldots, \lceil \log_{3/2} n \rceil$. At iteration i, one is given a distance matrix D, containing upper bounds on the distances between nodes. Initially D contains edge weights ($+\infty$ for missing edges). The algorithm samples a subset B_i of *bridge* nodes, where each node is sampled independently with probability $p_i = \min\{1, \frac{9 \ln n}{s_i}\}$, $s_i = (3/2)^i$. Then one sets,

$$D \leftarrow \min\{D, round_{s_i M}(D[V, B_i]) \star round_{s_i M}(D[B_i, V])\},$$

where the minimum is computed element-wise. Here $round_{M'}(\cdot)$ is a function that takes as input a matrix and returns the same matrix where the entries of absolute value larger than M' are set to $+\infty$. Zwick shows that for any i and any two nodes u, v, if there is a shortest path from u to v on at most $(3/2)^i$ edges, then after iteration i w.h.p. $D[u, v] = dist(u, v)$. By Lemma 2.1, the runtime of the algorithm up until some given iteration ℓ is $\tilde{O}(Mn^\omega s_\ell^{3-\omega})$, and after that iteration is $\tilde{O}(n^3/s_\ell)$: hence the overall runtime is $\tilde{O}(M^{1/(4-\omega)} n^{2+1/(4-\omega)})$. At the end of the algorithm, the shortest path distances are given by D with probability at least $1 - 1/n$. It is not hard to show that, by replacing the factor 9 in the probability p_i with $3Q + 6$ for $Q > 1$, the failure probability decreases to n^{-Q}: we next consider this variant of the algorithm. By halting the algorithm after ℓ iterations, we obtain the following corollary that we will need later.

Corollary 3.1. *The distances between all pairs of nodes that have shortest paths on at most S nodes can be computed in time $\tilde{O}(Mn^\omega S^{3-\omega})$, with failure probability polynomially small in n.*

We next adapt Zwick's algorithm to solve STSP, by making the following changes:

(1) Starting from $B_0 = V$, we let B_i be a random subset of B_{i-1} so that $|B_i| = p_i \cdot |V|$ where as before $p_i = \min\{1, (9 \ln n)/s_i\}$.
(2) We update only a portion of the matrix D at each iteration according to the formula

$$D[S \cup B_i, T \cup B_i] \leftarrow \min\{D[S \cup B_i, T \cup B_i], \\ round_{s_i M}(D[S \cup B_i, B_i]) \star round_{s_i M}(D[B_i, T \cup B_i])\}.$$

At the end of the process the submatrix $D[S, T]$ contains the desired distances. The first change introduces a dependency between the sets B_i at different iterations, which is crucial for our purposes.

This type of dependency was also used by Yuster and Zwick [47] in the construction of their distance oracle. The second step allows us to save time, while computing distances for the relevant pairs of nodes as in Zwick's original algorithm. This step is where we improve on the runtime for STSP obtained by applying the distance oracle of [47] directly.

Reminder of Theorem 1.4. *There is a randomized algorithm that solves STSP in n-node directed graphs with integer weights in* $[-M, M]$ *in time* $\tilde{O}(Mn^{\omega} + |S| \cdot |T| \cdot (Mn)^{\frac{1}{4-\omega}})$*, with failure probability polynomially small in n.*

Proof. The correctness analysis follows along the same line as in [49] and [47]. Consider next the runtime. Let us assume $\sigma := |S| \leq |T| =: \tau$, the other case being symmetric. We also let $\gamma \leq \sigma$ be a proper threshold to be fixed later, and $\beta_i := |B_i|$. At a given iteration i, we need to compute the distance product of a $(\sigma + \beta_i) \times \beta_i$ matrix by a $\beta_i \times (\tau + \beta_i)$ matrix with entries (other than $+\infty$) of absolute value at most $\tilde{O}(Mn/\beta_i)$: this costs the minimum of $\tilde{O}((\sigma + \beta_i)\beta_i(\tau + \beta_i))$ and $\tilde{O}(\frac{Mn}{\beta_i^{3-\omega}}(\sigma + \beta_i)(\tau + \beta_i))$ by Lemma 2.1. For $\beta_i \geq \sigma$, this is at most $\tilde{O}(\frac{Mn^2}{\beta_i^{2-\omega}}) \leq \tilde{O}(Mn^{\omega})$. For $\sigma > \beta_i \geq \gamma$, this is at most $\tilde{O}(\frac{Mn}{\beta_i^{3-\omega}}\sigma\,\tau) \leq \tilde{O}(\frac{Mn}{\gamma^{3-\omega}}\sigma\,\tau)$. In the remaining case $\gamma > \beta_i$ the runtime is $\tilde{O}(\sigma\,\tau\,\beta_i) \leq \tilde{O}(\sigma\,\tau\,\gamma)$. Choosing $\gamma = (Mn)^{\frac{1}{4-\omega}}$ gives the claimed runtime. □

We will need the following corollary, which is similar in spirit to Corollary 3.1.

Corollary 3.2. *Given an n-node directed graph G with integer edge weights in* $[-M, M]$*, an integer* $1 \leq Z \leq n-1$*, and a subset W of q nodes with* $q = \tilde{O}(\frac{n}{Z})$*. In time* $\tilde{O}(Mn^{\omega})$ *one can compute upper bounds on the pairwise distances among nodes in W so that, with failure probability polynomially small in n, the distance is computed correctly whenever there exists a shortest path of hop-length at most Z.*

Proof. It is sufficient to truncate the execution of the algorithm from Theorem 1.4 as soon as $s_i \geq Z$. Modulo poly-logarithmic factors, the running time is

$$\sum_{i=1}^{\lceil \log_{3/2} Z \rceil} Ms_i \left(\frac{n}{s_i}\right)^{\omega} = Mn^{\omega} \sum_{i=1}^{\lceil \log_{3/2} Z \rceil} \left(\frac{1}{s_i}\right)^{\omega-1} = O(Mn^{\omega}).$$

□

4 REPLACEMENT PATHS

In this section we present our recursive RP algorithm `rp` running in time $\tilde{O}(Mn^{\omega})$. The main algorithm is described in Figure 3. In the pseudocode we assume that each variable corresponding to a graph also carries the associated edge weights.

The algorithm initially (lines 1-2) computes the shortest s-t path $P_{s,t} = P = (s = v_1, v_2, \ldots, v_l = t)$, as well as all the distances $dist_P(v_i, v_j)$ between pairs of nodes along P with $i < j$. Then (lines 3-4), it fills in a list L of triples of type (v_i, v_j, d), with $i < j$, where d is the length of some s-t path avoiding all edges along P between v_i and v_j. Finally (lines 6-15), the list L is processed as described in Section 1.2.1 to obtain all the replacement paths lengths.

We next focus on the recursive procedure `recRP`, which is the core of our algorithm. The procedure is described in Figure 4. Let Z be a sub-polynomial function of n to be fixed later. The input to the procedure is a pair $(\tilde{G}, \tilde{P})$. Here $\tilde{G}$ is an edge-weighted multi-digraph with $\tilde{n}$ nodes and largest absolute weight $\tilde{M}$ containing nodes s and t. Furthermore, $\tilde{P}$ is a path in $\tilde{G}$ from $\tilde{s}$ to $\tilde{t}$ that is also a subpath of P. Initially $\tilde{G} = G$ and $\tilde{P} = P$. As it will be clearer later, by construction there are at most $O(\log_Z n)$ parallel edges between each pair of nodes in $\tilde{G}$. As standard, in order to compute distances in $\tilde{G}$ or in its subgraphs, it is sufficient to keep a minimum length edge for each

Figure 3 Algorithm rp for RP. The input to the problem is (G, s, t). Here L , P, s, t, and $dist_P(\cdot,\cdot)$ are considered as global variables used by Procedure recRP.

1: Find the shortest s-t path $P = (s = v_1 \to v_2 \to \ldots, v_l = t)$
2: Find $\forall v_i, v_j \in P$ their distance $dist_P(v_i, v_j)$ in P
3: Set $L \leftarrow \emptyset$; Initialize $\tilde{D}_{s,t,e} \leftarrow +\infty$ for all $e \in E(P)$
4: Call recRP(G, P)
5: Sort all elements $(v_j, v_k, d) \in L$ in nondecreasing order of d
6: Store every (v_i, i) in a binary search tree T with search key i
7: **while** $L \neq \emptyset$ **do**
8: Pop (v_j, v_k, d) from L
9: Find the successor v_q of v_j in T
10: **for** every $(v_p, p) \in T$ with $q \leq p \leq k-1$ **do**
11: Set $\tilde{D}_{s,t,v_p v_{p+1}} \leftarrow d$
12: Remove (v_p, p) from T
13: **end for**
14: **end while**
15: Output $\{\tilde{D}_{s,t,e}\}_{e \in E(P)}$

set of parallel (equally directed) edges. This does not affect the asymptotic runtime. We assume that parallel edges[16] are labeled so that we are able to identify edges corresponding to the original path P.

The goal of the procedure is to compute all the lengths of replacement paths for edges in $E(\tilde{P})$, and add a corresponding entry to L. Recall that $W(\tilde{P}) := V(\tilde{P}) \setminus \{\tilde{s}, \tilde{t}\}$. We can classify the detours of such replacement paths in four types according to their starting node $\tilde{v}$ and ending node $\tilde{u}$:

(1) $\tilde{v}, \tilde{u} \notin W(\tilde{P})$;
(2) $\tilde{v} \notin W(\tilde{P})$ and $\tilde{u} \in W(\tilde{P})$;
(3) $\tilde{v} \in W(\tilde{P})$ and $\tilde{u} \notin W(\tilde{P})$;
(4) $\tilde{v}, \tilde{u} \in W(\tilde{P})$.

The replacement paths of the first three types are handled in lines 1-9. The procedure initially computes the graph $\tilde{G}_{\tilde{P}}$ as described in Section 1.2.1. Recall that in this graph we remove the edges of $\tilde{P}$ and split all nodes $v_i \in W(\tilde{P})$ into v_i^{in} (with no outgoing edges) and v_i^{out} (with no incoming edges). The shortest s-t path in $\tilde{G}_{\tilde{P}}$ gives the best replacement path of Type (1). The replacement paths of Type (2) are handled in lines 4-6. Any such path corresponds to a shortest path from s to some v_i^{in}, $v_i \in W(\tilde{P})$, in $\tilde{G}_{\tilde{P}}$ plus the portion of P from v_i to t. The replacement paths of Type (3) are handled symmetrically in lines 7-9. In each case we add a proper triple $(\tilde{v}, \tilde{u}, d)$ to L, so that the corresponding replacement path lengths are computed correctly in lines 5-15 of rp.

It remains to consider replacement paths of Type (4), which are handled recursively. The base of the recursion (lines 10-15) is when $\tilde{n} \leq Z$. In this case we use a brute force approach: We simply compute the shortest paths between all pair of nodes in $W(\tilde{P})$ in the graph $\tilde{G} - E(\tilde{P})$, and from that we derive the corresponding entries of L.

The recursive case (lines 16–27) works as follows. First of all, we partition $\tilde{P}$ into Z subpaths $P_1, \ldots, P_Z$ of roughly the same length (i.e., of length $\lfloor |E(\tilde{P})|/Z \rfloor$ or $\lceil |E(\tilde{P})|/Z \rceil$). This is similar to the bucketting algorithm described in Section 1.2.1, however here Z is a sub-polynomial (rather than polynomial) function of n. For each subpath P_i we perform the following steps. First of all, we sample a set B_i of $\Theta(\frac{\tilde{n}}{Z} \cdot \log n)$ nodes uniformly at random as in Corollary 2.6 (where N, n,

[16]It is possible to avoid parallel edges, but the algorithm and analysis become more technical.

Figure 4 Procedure recRP. The input is a pair $(\tilde{G}, \tilde{P})$. By $\tilde{n}$ and $\tilde{M}$ we denote the number of nodes and the largest absolute weight in $\tilde{G}$, respectively. The starting and ending nodes of $\tilde{P}$ are denote by $\tilde{s}$ and $\tilde{t}$, respectively. Here Z is a sub-polynomial function of n, while L, P, s, t, and $dist_P(\cdot,\cdot)$ are considered as global variables.

1: Compute graph $\tilde{G}_{\tilde{P}}$ as described in Section 1.2.1
2: Compute distances $dist_{\tilde{G}_{\tilde{P}}}(s,\cdot)$ and $dist_{\tilde{G}_{\tilde{P}}}(\cdot,t)$ using Corollary 2.4
3: Add $(\tilde{s},\tilde{t},dist_{\tilde{G}_{\tilde{P}}}(s,t))$ to L
4: **for** all nodes of type $v_i^{in} \in V(\tilde{G}_{\tilde{P}})$ **do**
5: Add $(\tilde{s},v_i,dist_{\tilde{G}_{\tilde{P}}}(s,v_i^{in}) + dist_P(v_i,t))$ to L
6: **end for**
7: **for** all nodes of type $v_i^{out} \in V(\tilde{G}_{\tilde{P}})$ **do**
8: Add $(v_i,\tilde{t},dist_P(s,v_i) + dist_{\tilde{G}_{\tilde{P}}}(v_i^{out},t))$ to L
9: **end for**
10: **if** $\tilde{n} \le Z$ **then**
11: Compute distances $dist_{\tilde{G}-E(\tilde{P})}(\cdot,\cdot)$ between all pairs of nodes in $W(\tilde{P})$
12: **for** every $v_i, v_j \in W(\tilde{P}), i < j$ **do**
13: Add $(v_i,v_j,dist_P(s,v_i) + dist_{\tilde{G}-E(\tilde{P})}(v_i,v_j) + dist_P(v_j,t))$ to L
14: **end for**
15: **else**
16: Partition $\tilde{P}$ into Z subpaths $P_1,\ldots,P_Z$ with roughly the same number of edges
17: **for** $i = 1,\ldots,Z$ **do**
18: Sample $\Theta(\frac{\tilde{n}}{Z}\log n)$ nodes B_i uniformly at random as in Corollary 2.6
19: Construct a complete digraph G_i on node set $V_i := V(P_i) \cup B_i \cup \{s,t\}$ with edge weights w_i initialized to $+\infty$.
20: Compute estimates $dist'(\cdot,\cdot)$ of pairwise distances among nodes V_i in $\tilde{G} - E(P_i)$ using Corollary 3.2 with parameter Z.
21: **for** every $uv \in E(G_i)$ **do**
22: **if** $|dist'(u,v)| \le Z\tilde{M}$ **then**
23: $w_i(uv) \leftarrow dist'(u,v)$
24: **end if**
25: **end for**
26: Add $E(P_i)$, with the associated weights, to G_i .
27: Call recRP(G_i,P_i)
28: **end for**
29: **end if**

and n' are replaced by n, $\tilde{n}$, and $2\tilde{n}/Z$, respectively). We construct a complete digraph G_i on nodes $V_i := V(P_i) \cup B_i \cup \{s,t\}$. Next we compute the distances in $\tilde{G} - E(P_i)$ between pairs of nodes in V_i. We label each edge of G_i with the corresponding distance, if its absolute value is at most $Z\tilde{M}$, and otherwise we label it with $+\infty$. At this point we add back the edges of $E(P_i)$. Finally, we apply the procedure recursively to the pair (G_i,P_i).

We first consider the runtime of rp

Lemma 4.1. *Procedure* rp *runs in* $\tilde{O}(Mn^{\omega})$ *time.*

Proof. Line 1 can be executed in time $\tilde{O}(Mn^{\omega})$ using Corollary 2.4. Line 2 can be easily implemented in $O(n^2)$ time.

We next consider how many triples are ever added by the algorithm to L. We will show that that number is $O(n^2)$, and hence lines 7-15 of Algorithm rp can be executed in $\tilde{O}(n^2)$ time.

The depth of the recursion is $O(\log_Z n) \leq O(\log n)$, and the number of generated subproblems in recursion level p is at most Z^p. Line 3 of recRP adds at most one entry to L for each subproblem. Observe that the number of such subproblems is at most $O(n)$. Indeed, consider the recursion tree (that has one node per subproblem). The leaves of such tree induce a partition of the edges of the input path, hence there are at most n many such leaves. The internal nodes of the recursion tree have degree $Z \geq 2$, hence their number is upper bounded by the number of leaves. Each node v_i^{in} is considered at most $O(\log n)$ times in line 4, hence line 5 adds at most $O(n \log n)$ entries to L altogether. The same holds for line 8. The total number of pairs (v_i, v_j) considered in line 13 in recursion level p is $O(Z^p(n/Z^p)^2) = O(n^2/Z^p)$, and summing over all levels, we get that $O(n^2)$ triples are added to L by line 13. Finally, each edge $v_j v_{j+1}$ is considered at most $O(\log n)$ times in line 23, hence the total number of entries added to L in line 24 is $O(n \log n)$. The claim follows.

We next analyze the runtime of the calls to recRP. Consider some level $p = 0, \ldots, O(\log_Z n)$ of the recursion tree. At this level the algorithm generates at most Z^p instances. Consider one such instance $(\tilde{G}, \tilde{P})$. Observe that $\tilde{G}$ and $\tilde{P}$ contain $\tilde{n} = \Theta(n/Z^p \cdot log^p n)$ and $\Theta(n/Z^p)$ nodes, respectively. Furthermore, the largest absolute weight in $\tilde{G}$ is $\tilde{M} \leq MZ^p$. Intuitively, the increase of the value of $\tilde{M}$ in the subproblems is compensated by a comparable decrease of the number $\tilde{n}$ of considered nodes.

The total runtime of lines 1-9 is dominated by the execution of line 2, which can be performed in time $\tilde{O}(\tilde{M}\tilde{n}^\omega))$ by Corollary 2.4. Consider next lines 17–27 (excluding the time spent on recursive calls). The algorithm generates Z subproblems. The time spent on each subproblem (lines 19-27) is dominated by the distances computation of line 20, which can be performed in time $\tilde{O}(\tilde{M}\tilde{n}^\omega)$ by Corollary 3.2.

Altogether each instance of level p requires time $\tilde{O}(Z\tilde{M}\tilde{n}^\omega) = \tilde{O}(Z^{p+1}M\left(\frac{n \log^p n}{Z^p}\right)^\omega)$. Since there are Z^p such instances, the total runtime on level p is

$$\tilde{O}(ZMn^\omega \frac{\log^{p\omega} n}{Z^{p(\omega-2)}}).$$

We can guarantee that lines 1-9 plus 17–27 at level p in the recursion (hence altogether) take time $\tilde{O}(Mn^\omega)$ by imposing the following conditions on Z: for $\omega > 2$, set $Z = 2\log^{\omega/(\omega-2)} n$, and for $\omega = 2$, choose Z so that $Z = n^{o(1)}$ and $(\log n)^{\log_Z n} = n^{o(1)}$; e.g. $Z = 2^{C\sqrt{\log n}}$ works.

It remains to consider the total time spent on base cases (lines 10-15). There are $O(n/Z)$ base case instances, each one involving at most Z nodes. We can compute all-pairs distances trivially in $O(Z^3)$ time in each such instance. Hence, the total runtime of solving these instances is $O(nZ^2) = \tilde{O}(n)$. The claim follows. □

We next discuss the correctness of rp.

Lemma 4.2. *For every pair $(\tilde{G}, \tilde{P})$ on which we execute* recRP*, the distances between nodes in $V(\tilde{G})$ are the same in $G - E(\tilde{P})$ and in $\tilde{G} - E(\tilde{P})$ w.h.p.*

Proof. Let us assume that the high probability events of Corollaries 2.6 and 3.2 hold whenever we execute lines 18 and 20. The claim on the probability then follows by the union bound for a sufficiently large constant C, observing that the total number of recursive calls is polynomially bounded.

We prove the claim by induction. The base case is when $(\tilde{G}, \tilde{P}) = (G, P)$, where the claim trivially holds. Next assume that the claim holds for a given pair $(\tilde{G}, \tilde{P})$. It is sufficient to show that it then holds also for any associated subproblem (G_i, P_i). By assumption any shortest path in $\tilde{G} - E(P_i)$ between nodes in $\{s, t\} \cup V(P_i)$ is partitioned by B_i into subpaths of at most Z edges (hence of

absolute value at most $Z\tilde{M}$). Therefore $G_i - E(P_i)$ preserves the distances of $\tilde{G} - E(P_i)$ between nodes $V_i = V(G_i)$. Thus by inductive hypothesis the same holds w.r.t. to $G - E(P_i)$. □

Lemma 4.3. *Procedure* rp *solves RP with polynomially small failure probability in n.*

Proof. Let us assume that the high-probability event of Lemma 4.2 holds. Consider a given edge $e \in E(P)$, and the associated replacement path of length d. Let the corresponding detour start at node $\tilde{v}$ and end at node $\tilde{u}$. By the previous discussion it is sufficient to show that a triple $(\tilde{v}, \tilde{u}, d)$ is added to L at some point. Let recRP$(\tilde{G}, \tilde{P})$ be the lowest level call of recRP where both $\tilde{v}$ and $\tilde{u}$ belong to $\tilde{P}$. If the considered call is a base case, a proper triple is added to L in lines 10-15. Otherwise, the considered detour must correspond to one of the cases (1), (2), and (3) described before, and a corresponding triple is added to L in lines 1-9. □

We are now ready to prove Theorem 1.1.

Reminder of Theorem 1.1. *There is a randomized algorithm that solves RP in n-node directed graphs with integer weights in $[-M, M]$ in $\tilde{O}(Mn^{\omega})$ time, with failure probability polynomially small in n.*

Proof. Consider algorithm rp. Its runtime is $\tilde{O}(Mn^{\omega})$ by Lemma 4.1. This algorithm is correct with high probability by Lemma 4.3. The claim follows. □

5 SINGLE-SOURCE REPLACEMENT PATHS

We show how to solve SSRP, by reducing it to a small (subpolynomial) set of subpath problems and smaller instances (Section 5.1), and by solving each subpath problem efficiently (Section 5.2).

Recall that in SSRP we are given as input a graph G with edge weights $w(\cdot)$ and a source node s. For any $t \in V(G)$ and for any edge e along the shortest path $P_{s,t}$ from s to t, we wish to compute the length $D_{s,t,e}$ of the replacement path for the triple (s, t, e).

The shortest path tree T_s and the corresponding distances $dist_G(s, \cdot)$ can be computed in $O(Mn^{\omega})$ time by Corollary 2.4. If u is a descendant of v in T_s, one can derive in constant time the values $dist_G(v, u) = dist_G(s, u) - dist_G(s, v)$. Hence we will assume that the latter quantities are implicitly given. Our estimates $\tilde{D}_{s,t,e}$ of $D_{s,t,e}$ are initialized to $+\infty$, and considered as global variables as well.

By Z we will denote a suitably chosen subpolynomial function of n, which is a global variable. Similarly to the RP case, for $\omega > 2$, one can set $Z = \text{poly}\log(n)$ and otherwise (say) $Z = 2^{\sqrt{\log n \log\log n}}$.

5.1 From SSRP to Subpath Problems

We next describe a recursive procedure ssrp to solve SSRP. This procedure takes as input a pair $(\tilde{G}, \tilde{T})$. Here $\tilde{G}$ is an $\tilde{n}$-node multi-digraph $\tilde{G}$ with edge weights $\tilde{w}(\cdot)$ of absolute value at most $\tilde{M}$, containing the source node s of the starting input instance. We will guarantee that there are at most $O(\log_Z n)$ parallel edges at any time, hence the cost of handling such parallel edges is negligible analogously to the RP case. Furthermore $\tilde{T}$ is a subtree of $\tilde{G}$ and of T_s, rooted at the node $\tilde{t}$ in $\tilde{T}$ at minimum hop-distance from s in T_s. By $\tilde{P}$ we will denote the shortest path from s to $\tilde{t}$.

The goal of the procedure is to compute $D_{s,t,e}$ for all $(t, e) \in V(\tilde{T}) \times E(\tilde{T})$[17]. It is then sufficient to call this procedure on input (G, T_s).

We remark that by n we denote the number of nodes in the input graph G: we will guarantee that the failure probability of ssrp is polynomially small in n (and hence the overall algorithm fails with polynomially small probability by the union bound).

[17] Whenever $e \notin E(P_{st})$, we implicitly assume that the pair (t, e) is neglected. This is to slightly simplify the notation.

Figure 5 Recursive algorithm ssrp for SSRP. The input is a pair $(\tilde{G}, \tilde{T})$. Here $\tilde{n}$ and $\tilde{M}$ are the number of nodes and largest absolute weight in $\tilde{G}$, respectively. As usual the weight function of $\tilde{G}$ is considered implicit. Tree $\tilde{T}$, rooted at $\tilde{t}$, is a subtree of T_s. Here Z is a subpolynomial function of the original number of nodes n, and $C > 0$ is a sufficiently large constant.

1: **if** $\tilde{n} < Z$ **then**
2: Compute with the trivial cubic algorithm all $D_{s,t,e}$ values w.r.t. $\tilde{G}$, $(t, e) \in V(\tilde{T}) \times E(\tilde{T})$, and set $\tilde{D}_{s,t,e} \leftarrow \min\{\tilde{D}_{s,t,e}, D_{s,t,e}\}$
3: **return**
4: **end if**
5: Using balanced tree separators, partition $\tilde{T}$ into subtrees $T_1, \ldots, T_{Z'}$, $Z' = \Theta(Z)$, each one with $\Theta(\frac{\tilde{n}}{Z})$ nodes.
6: **for** i=1,…,Z' **do**
7: Call subpath$(\tilde{G}, T_i)$
8: Sample $\frac{\tilde{n}}{Z} \cdot C \log n$ nodes B_i uniformly at random
9: Construct a complete digraph G_i on node set $V_i := V(T_i) \cup B_i \cup \{s\}$ with edge weights w_i initialized to $+\infty$.
10: Compute estimates $dist'(\cdot, \cdot)$ of pairwise distances among nodes V_i in $\tilde{G} - E(T_i)$ using Corollary 3.2 with parameter Z.
11: **for** every $uv \in E(G_i)$ **do**
12: **if** $|dist_{\tilde{G}-E(T_i)}(u, v)| \leq Z\tilde{M}$ **then**
13: $w_i(uv) \leftarrow dist_{\tilde{G}-E(T_i)}(u, v)$
14: **end if**
15: **end for**
16: Add $E(T_i)$, with the associated weights, to G_i .
17: Call ssrp(G_i, T_i)
18: **end for**

We consider Algorithm ssrp in Figure 5. Steps 1-4 address the base case, where the problem is solved with the trivial cubic algorithm for small enough $\tilde{n}$.

Otherwise, in Step 5 we partition $\tilde{T}$ into $Z' = \Theta(Z)$ subtrees with roughly the same number of nodes $\Theta(\frac{\tilde{n}}{Z})$ using balanced tree separators. In more detail, recall that for each tree T of n nodes there exists a node v (balanced tree separator) such that T can be split into two edge-disjoint trees T' and T'' sharing node v only, and each one containing at least $n/3 + 1$ nodes. By applying this procedure recursively it is easy to obtain $Z' \in [Z, 3Z]$ trees each one containing between $\frac{\tilde{n}}{3Z}$ and $\frac{\tilde{n}}{Z}$ nodes. Each splitting step, and hence the overall procedure, can be performed in $\tilde{O}(\tilde{n})$ time.

Then for each tree T_i the algorithm works as follows. Let P_i be the shortest path in $\tilde{G}$ between s and the root t_i of T_i. For each $t \in V(T_i)$, the relevant pairs (t, e) are of the following two types: (a) $e \in E(P_i)$ or (b) $e \in E(T_i)$. We define the subproblem of computing $D_{s,t,e}$ for the pairs (t, e) of the first and second type a *subpath problem* and a *subtree problem*, respectively.

The subpath problem associated with T_i is solved by calling in Step 7 procedure subpath which is described in next subsection.

The subtree problem associated with T_i is instead addressed recursively in Steps 8-17. In more detail, we start by building a compressed multi-digraph G_i analogously to the RP case in Steps 8-16. Each such graph contains $\tilde{O}(\frac{\tilde{n}}{Z})$ nodes and has edge weights of absolute value at most $Z\tilde{M}$. By a similar argument as in the RP case, distances between nodes of G_i are the same in $G_i - E(T_i)$ and in $\tilde{G} - E(T_i)$ (hence in $G - E(T_i)$ by induction) w.h.p. Then in Step 17 we call recursively ssrp on (G_i, T_i).

Given the above algorithm, we can prove the following lemma.

Lemma 5.1. *Given an algorithm that solves a subpath problem in time $\tilde{O}(\tilde{M}^\alpha \tilde{n}^\beta)$ with failure probability at most n^{-Q}, for constants $Q > 0$ and $\beta \geq \alpha + 1 \geq 1$, there is an algorithm that solves SSRP in time $\tilde{O}(Mn^\omega + M^\alpha n^\beta)$ with failure probability at most $\tilde{O}(n^{-Q+1})$.*

Proof. By the union bound the failure probability of the algorithm can be upper-bounded by summing the failure probabilities of the $\tilde{O}(n)$ subpath problems and the $\tilde{O}(n)$ compression steps. Analogously to the RP case, the compression step in each subtree problem preserves the correct replacement path distances with a failure probability that can be made as small as n^{-Q} by choosing properly the constants in the sampling step and in the computation of shortest paths. The claim follows.

Now consider the runtime. In each recursive call involving $\tilde{n}$ nodes and absolute weights at most $\tilde{M}$, we partition the tree into at most $3Z$ subtrees containing at most $\tilde{n}/Z$ nodes each, and then we perform a compression step that increases the number of nodes to at most $2C\tilde{n}\log n/Z$ and the absolute weights to at most $\tilde{M}Z$. Thus at level $i \geq 0$ of the recursion tree the algorithm executes at most $(3Z)^{i+1}$ compression steps and subpath procedures on instances on at most $n(2C\log n)^i/Z^i$ nodes and with weights of absolute value at most MZ^i. The number of recursive levels is at most $\log_{Z/(2C\log n)} n$. The (leaf) instances on at most Z nodes are solved using the cubic time algorithm in overall time $O(nZ^3) = \tilde{O}(n)$. We analyze the rest of the runtime.

Consider first the compression steps. Each compression step can be performed in time $\tilde{M}\tilde{n}^\omega g(n)$ for some subpolynomial function $g(\cdot)$. Hence, all the compression steps generated by SSRP instances at level i in the recursion tree can be performed in time

$$(3Z)^{i+1} \cdot (MZ^i)\left(n(\frac{2C\log n}{Z})^i\right)^\omega g(n) \overset{\omega\geq 2}{\leq} Mn^\omega 3Z\, g(n) \cdot (6C\log n)^{i\,\omega}.$$

Summing over all the levels i of the recursion tree, the runtime of the compression steps is

$$\begin{aligned} &Mn^\omega 3Z\, g(n) \sum_{i=0}^{\log_{Z/(2C\log n)} n} (6C\log n)^{i\omega} \\ \leq &Mn^\omega 3Z\, g(n)\log n \cdot 2^{\omega \log(6C\log n)\frac{\log n}{\log Z - \log(2C\log n)}} \\ \overset{Z=2^{\sqrt{\log n \log\log n}}}{\leq} &Mn^\omega g(n) \cdot 2^{O(\sqrt{\log n \log\log n})}. \end{aligned}$$

In the first inequality above we simply upper bounded the sum with $1 + \log_{Z/(2C\log n)} n \leq \log n$ times the largest term in the sum, which is achieved for the largest value of i.

Similarly, the subpath problems generated by SSRP instances at level $i \geq 0$ in the recursion tree take time at most

$$(3Z)^{i+1} \cdot (MZ^i)^\alpha \left(n(\frac{2C\log n}{Z})^i\right)^\beta g(n) \overset{\beta\geq\alpha+1}{\leq} M^\alpha n^\beta 3Z\, g(n) \cdot (6C\log n)^{i\beta},$$

and hence their total execution time is at most

$$M^\alpha n^\beta 3Z\, g(n)\log n \cdot 2^{\beta\log(6C\log n)\frac{\log n}{\log Z - \log(2C\log n)}} \leq M^\alpha n^\beta g(n) \cdot 2^{O(\sqrt{\log n \log\log n})}.$$

The claim on the runtime follows. □

5.2 Solving Subpath Problems

Consider a subpath problem $(\tilde{G}, \tilde{T})$, where as usual $\tilde{G}$ has $\tilde{n}$ nodes and largest absolute weight $\tilde{M}$. By $\tilde{P}$ we denote the shortest path in $\tilde{G}$ between s and the root $\tilde{t}$ of $\tilde{T}$. Recall that our goal is to compute $D_{s,t,e}$ for all $(t,e) \in V(\tilde{T}) \times E(\tilde{P})$. We will show how to do that in time $\tilde{O}(\tilde{M}\tilde{n}^\omega)$. W.l.o.g. we can

Figure 6 Procedure subpath to solve a subpath problem $(\tilde{G}, \tilde{T})$. Here $\tilde{G}$ has $\tilde{n}$ nodes and largest absolute weight $\tilde{M}$. By $\tilde{P} = (v_1, v_2, \ldots, v_h)$ we denote the shortest path in $\tilde{G}$ from s to the root $\tilde{t}$ of $\tilde{T}$, and by V_x the last x nodes of that path. Here L is a proper integer parameter. In the case of arbitrary weights Steps 3-9 are replaced by $\tilde{D}^{detour}_{v,t} \leftarrow dist_{\tilde{G}-E(\tilde{P})}(v,t)$ for any $(v,t) \in V(\tilde{P}) \times V(\tilde{T})$, where such distances are computed with the algorithm from Theorem 2.2.

1: Solve the RP problem induced by $\tilde{P}$ using the algorithm from Theorem 1.1, and let $\tilde{D}_{s,\tilde{t},e}$ be the obtained estimates of $D_{s,\tilde{t},e}$.
2: Set $\tilde{D}_{s,t,e} \leftarrow \min\{\tilde{D}_{s,t,e}, \tilde{D}_{s,\tilde{t},e} + dist_G(\tilde{t},t)\}$ for all $(t,e) \in V(\tilde{T}) \times E(\tilde{T})$.
3: Initialize $\{\tilde{D}^{detour}_{v,t}\}_{(v,t)\in V(\tilde{P})\times V(\tilde{T})}$ to $+\infty$ and let $G' := \tilde{G} - E(\tilde{P})$.
4: Compute $dist_{G'}(v,t)$ for all $(v,t) \in V_{\tilde{ML}} \times V(\tilde{T})$ using the STSP algorithm from Theorem 1.4, and set $\tilde{D}^{detour}_{v,t} \leftarrow dist_{G'}(v,t)$
5: **for** $i = 0, \ldots, \lceil \log_2 \frac{\tilde{n}}{L} \rceil$ **do**
6: Sample $\tilde{n}/X_i \cdot C \log n$ nodes B_i, where $X_i = 2^i L$.
7: Compute $dist_{G'}(b,v)$ and $dist_{G'}(v,b)$ for all $b \in B_i$ and $v \in V$ using the STSP algorithm from Theorem 1.4.
8: Set $\tilde{D}^{detour}_{v,t} \leftarrow \min\{\tilde{D}^{detour}_{v,t}, dist_{G'}(v,b) + dist_{G'}(b,t)\}$ for every $(v,b,t) \in V_{2MX_i} \times B_i \times V(\tilde{T})$.
9: **end for**
10: **for** all $t \in V(\tilde{T})$ **do**
11: Set $d_1 \leftarrow \tilde{D}^{detour}_{s,t}$, and $d_i \leftarrow \min\{d_{i-1}, dist_G(s,v_i) + \tilde{D}^{detour}_{v_i,t}\}$ for $i = 2, \ldots, h-1$.
12: Set $\tilde{D}_{s,t,e_i} \leftarrow \min\{\tilde{D}_{s,t,e_i}, d_i\}$ for $i = 1, \ldots, h-1$ and $e_i := v_i v_{i+1}$.
13: **end for**

assume that $\tilde{M} \leq \tilde{n}^{3-\omega}$ since otherwise we can solve trivially the problem in $\tilde{O}(\tilde{n}^3) \subseteq \tilde{O}(\tilde{M}\tilde{n}^\omega)$ time.

We use procedure subpath described in Figure 6. As mentioned in the introduction, we distinguish between two types of replacement paths $P_{s,t,e}$ for the considered pairs (t,e), $e = uv$. A *jumping path* $P_{s,t,e}$ leaves $\tilde{P}$ at some node (between s and u) and then meets $\tilde{P}$ again at some other node (between v and $\tilde{t}$). A *departing path* $P_{s,t,e}$ leaves $\tilde{P}$ at some node (between s and u) and never meets $\tilde{P}$ again.

Jumping paths are addressed in Steps 1-2 via a simple reduction to RP: indeed in this case the considered replacement path for the triple (s,t,e) is a replacement path for the triple $(s,\tilde{t},e)$ followed by a shortest path from $\tilde{t}$ to t. Here we fix the parameters in the algorithm (without substantially affecting the runtime), so that the failure probability is polynomially small in n (rather than in $\tilde{n}$). Alternatively, it is sufficient to repeat Steps 1-2 for $\Theta(\log n / \log \tilde{n})$ many times.

It remains to consider the departing paths. Consider first the case of arbitrary weights. We start by observing that it is sufficient to compute all the distances $dist_{G'}(v,t)$ from nodes v in $\tilde{P}$ to nodes t in $\tilde{T}$ in the graph $G' := \tilde{G} - E(\tilde{P})$. Let $s = v_1, v_2 \ldots v_h = \tilde{t}$ be the sequence of nodes in $\tilde{P}$. For $e = v_i v_{i+1}$ and any $t \in V(\tilde{T})$, the shortest departing path for (s,t,e) has length $\min_{j\leq i}\{dist_G(s,v_j) + dist_{G'}(v_j,t)\}$. For a fixed t, we can compute these quantities for all $e \in \tilde{P}$ via a single scan of the nodes of $\tilde{P}$ from v_1 to v_h (updating the corresponding minimum each time). This takes $O(n^2)$ time, and is performed in Steps 10-13. For the computation of the distances $dist_{G'}(v,t)$ one can directly apply Zwick's APSP algorithm to graph G'.

Lemma 5.2. *There is an algorithm which solves a given subpath problem on an $\tilde{n}$-node directed graph with integer weights in $[-\tilde{M}, \tilde{M}]$ in time $\tilde{O}(\tilde{M}^{\frac{1}{4-\omega}}\tilde{n}^{2+\frac{1}{4-\omega}})$, with failure probability polynomially small in n.*

The part of Theorem 1.3 relative to negative weights follows from Lemmas 5.1 and 5.2.

The rest of this section is devoted to the computation of departing paths in the case of positive weights (Steps 3-9). Let V_x be the final x nodes of $\tilde{P}$. We first consider the detours (of departing paths) which contain at most L nodes, for a proper parameter L. As we already mentioned in the introduction, such detours must start at some node in $V_{\tilde{M}L}$: indeed, the length of these detours is at most $\tilde{M}(L-1)$, and this is also an upper bound on the length of the original shortest paths among the same endpoints; since weights are strictly positive integers, the latter paths cannot contain more than $\tilde{M}(L-1)+1 \leq \tilde{M}L$ nodes. Notice that such claim would not hold in the presence of negative (or even zero) weights.

We use the STSP algorithm from Theorem 1.4 to compute the distances $dist_{G'}(v,t)$ from $v \in V_{\tilde{M}L}$ to all $t \in V(\tilde{T})$ in G'. For the remaining detours, let us define $O(\log \tilde{n})$ intervals $[X_i, 2X_i)$ with $X_i = 2^i L$ and $0 \leq i \leq \lceil \log_2 \frac{\tilde{n}}{L} \rceil$. For each i, we search for detours with a number of nodes in $[X_i, 2X_i)$ as follows: we sample a bridge set B_i of $\frac{\tilde{n}}{X_i} \cdot C \log n$ nodes, so that B_i hits any detour on at least X_i nodes with polynomially (in n) small failure probability according to Corollary 2.6. We compute the distances in G' from any node in $V_{2\tilde{M}X_i}$ to any node in B_i and from any node in B_i to any node in $V(T_i)$, using our STSP algorithm from Theorem 1.4. For each $(v,t) \in V_{2\tilde{M}X_i} \times V(T_i)$, the desired distance is $\min_{b \in B_i}\{dist_{G'}(v,b) + dist_{G'}(b,t)\}$.

Lemma 5.3. *There is an algorithm which solves a given subpath problem on an $\tilde{n}$-node directed graph with integer weights in $[1, \tilde{M}]$ in time $\tilde{O}(\tilde{M}\tilde{n}^{\omega})$, with failure probability polynomially small in n.*

Proof. Consider the above algorithm. The algorithm fails if either the execution of the RP algorithm fails, or at least one of the executions of the STSP algorithm fails, or for some i the sample B_i does not hit all the detours on at least X_i nodes. The claim on the failure probability follows.

The computation of jumping paths and of departing paths from their detours takes $\tilde{O}(\tilde{M}\tilde{n}^{\omega})$ time. The computation of the detours themselves takes time

$$\tilde{O}(\tilde{M}\tilde{n}^{\omega} + \tilde{M}L\,\tilde{n}\,(\tilde{M}\tilde{n})^{\frac{1}{4-\omega}}) + \sum_i \tilde{O}(\tilde{M}\tilde{n}^{\omega} + \frac{\tilde{n}}{X_i}\tilde{n}(\tilde{M}\tilde{n})^{\frac{1}{4-\omega}} + \tilde{M}X_i\,\frac{\tilde{n}}{X_i}\,\tilde{n})$$

$$= \tilde{O}(\tilde{M}\tilde{n}^{\omega} + (\tilde{M}L\tilde{n} + \frac{\tilde{n}^2}{L})(\tilde{M}\tilde{n})^{\frac{1}{4-\omega}}).$$

Choosing $L = \sqrt{\tilde{n}/\tilde{M}}$ gives an overall runtime of $\tilde{O}(\tilde{M}\tilde{n}^{\omega} + \sqrt{\tilde{M}}\tilde{n}^{\frac{3}{2}}\,(\tilde{M}\tilde{n})^{\frac{1}{4-\omega}})$, which is $\tilde{O}(\tilde{M}\tilde{n}^{\omega})$ for any value of $\omega \in [2,3]$ since by assumption $\tilde{M} \leq \tilde{n}^{3-\omega} < \tilde{n}^{7-2\omega}$. □

Combining Lemmas 5.1 and 5.3, one obtains the part of Theorem 1.3 corresponding to positive weights (note that the compression step in procedure `ssrp` does not create negative weights if the input weights are positive).

6 DISTANCE SENSITIVITY ORACLES

In this section we present our improved DSOs. We start with the DSO from Theorem 1.5: we consider first the case of positive integer weights (Section 6.1), and later extend the result to allow for non-positive weights (Section 6.2). In Section 6.3 we describe the alternative DSO from Theorem 1.6. We conclude the section with a brief discussion about the space complexity.

6.1 Positive Weights

The basic strategy is as follows. Given two integer parameters $0 \leq S \leq L \leq n$, we distinguish 3 types of replacement paths: *hop-long* and *hop-short* replacement paths contain at least L and at most S nodes, respectively. The remaining paths are *hop-average*. We design a distinct oracle for each

kind of path. In particular, the oracle for hop-long paths will crucially exploit our SSRP algorithm. The preprocessing and query time of the overall oracle is given by the sum of the preprocessing and query times of these three oracles.

(1) Hop-short paths. We sample $S \cdot C \log n$ random graphs $G_1, \ldots, G_{S \cdot C \log n}$ as in Lemma 2.7. We compute all-pairs shortest paths on at most S nodes in each G_i as in Corollary 3.1, in time $\tilde{O}(S^{3-\omega}Mn^{\omega})$ per graph, and hence $\tilde{O}(S^{4-\omega}Mn^{\omega})$ altogether. For a query (s,t,e), it is sufficient to return the shortest distance from s to t in the graphs G_i not containing e. By Lemma 2.7 w.h.p. the number of considered graphs (and hence the query time) is $O(\log n)$, and at least one of them contains $P_{s,t,e}$ if it is hop-short.

(2) Hop-average paths. We sample $L \cdot C \log n$ random graphs $G_1, \ldots, G_{L \cdot C \log n}$ as in Lemma 2.7. We apply the preprocessing step of the distance oracle from Lemma 2.3 to each sampled graph. This takes $\tilde{O}(LMn^{\omega})$ preprocessing time, and allows us to answer a query (s,t,e), by considering all the $\Theta(\log n)$ graphs G_i not containing e, and querying the corresponding distance oracles in $\tilde{O}(n/S)$ time. By Lemmas 2.7 and 2.3, w.h.p. the answer is correct if $P_{s,t,e}$ is hop-average.

(3) Hop-long paths. We sample $\frac{n}{L} \cdot C \log n$ nodes B as in Lemma 2.5, so that w.h.p. B hits all the replacement paths on at least L nodes. We solve SSRP from any source $b \in B$ both in the original graph and in the graph where we reverse all the edges. The preprocessing time is $\tilde{O}(\frac{n}{L}Mn^{\omega})$. In order to answer a query (s,t,e), it is sufficient to consider the concatenation of replacement paths $P_{s,b,e}$ and $P_{b,t,e}$ for any $b \in B$: this takes $\tilde{O}(n/L)$ time, and returns the correct answer w.h.p. if $P_{s,t,e}$ is hop-long.

Altogether we obtain an $\tilde{O}(Mn^{\omega}(S^{4-\omega} + L + \frac{n}{L}))$ preprocessing time, and a $\tilde{O}(\frac{n}{S})$ query time. Setting $L = \Theta(\max\{\sqrt{n}, S\})$ concludes the proof of Theorem 1.5 for positive weights.

6.2 Negative Weights

We use the same approach as above for hop-short and hop-average paths (which also works in the presence of non-positive weights). For hop-long paths, we exploit a variant of our SSRP algorithm, where we are only interested in computing correctly the replacement paths on at most X nodes. Observe that, in each subpath problem (P', T'), it is sufficient to consider the detours of departing paths that start in the first X nodes; otherwise, the departing replacement path would be too long. Using our STSP algorithm, the runtime reduces to $\tilde{O}(Mn^{\omega} + XM^{\frac{1}{4-\omega}}n^{1+\frac{1}{4-\omega}})$. Note that we can use the same parameter X also in the recursive calls, since the compression step can only reduce the number of nodes in each path. By the same argument as in Lemma 5.1, this also upper-bounds the overall runtime of the algorithm.

Lemma 6.1. *For any $0 \leq X \leq n$, there is an algorithm of runtime $\tilde{O}(Mn^{\omega} + XM^{\frac{1}{4-\omega}}n^{1+\frac{1}{4-\omega}})$ for SSRP which computes correctly all the replacement path distances of paths with at most X nodes with failure probability polynomially small in n.*

We exploit the modified SSRP algorithm as follows. We define $O(\log n)$ intervals $[X_i, 2X_i)$ with $L \leq X_i := 2^i L < 2n$. In order to compute the replacement paths with a number of nodes in $[X_i, 2X_i)$, we sample $\frac{n}{X_i} \cdot C \log n$ nodes B_i as in Lemma 2.5, so that w.h.p. B_i hits all the replacement paths on at least X_i nodes. We solve SSRP from each source $b \in B_i$ in the original graph and in the graph with reversed edge directions, using the modified SSRP algorithm with parameter $X = 2X_i$. The preprocessing time is $\tilde{O}(\frac{n}{X_i}Mn^{\omega} + \frac{n}{X_i}X_i M^{\frac{1}{4-\omega}}n^{1+\frac{1}{4-\omega}}) \leq \tilde{O}(\frac{n}{L}Mn^{\omega} + M^{\frac{1}{4-\omega}}n^{2+\frac{1}{4-\omega}})$. For a query (s,t,e), it is sufficient to consider all the triples (s,b,t) with $b \in B_i$, which takes $\tilde{O}(\frac{n}{X_i}) \leq \tilde{O}(\frac{n}{L})$ time. Since $LMn^{\omega} + \frac{n}{L}Mn^{\omega} \geq Mn^{\omega+\frac{1}{2}} \geq M^{\frac{1}{4-\omega}}n^{2+\frac{1}{4-\omega}}$ for any $\omega \in [2,3]$, the extra term $M^{\frac{1}{4-\omega}}n^{2+\frac{1}{4-\omega}}$ is irrelevant in the runtime of the preprocessing stage. Hence we obtain (modulo polylogarithmic

factors) the same preprocessing and query time as in the case of positive weights. This concludes the proof of Theorem 1.5.

6.3 An Alternative Oracle

We next describe the DSO from Theorem 1.6. We again distinguish between hop-short, hop-average, and hop-long paths. We handle the first two types of paths as we did before. This takes $\tilde{O}(Mn^{\omega}(S^{4-\omega}+L))$ preprocessing time and $\tilde{O}(\frac{n}{S})$ query time.

Consider next hop-long paths. We exploit the $\tilde{O}(L)$ random graphs G_i that we used in the computation of hop-average paths. Recall that we precomputed the distance oracle from Lemma 2.3 for each such graph. We sample $\frac{n}{L} \cdot C \log n$ nodes B as in Lemma 2.5, and we compute all the distances of absolute value at most ML between pairs of nodes in B in each G_i. This can be done in $\tilde{O}(Mn^{\omega})$ time per graph as observed in [44]. We also construct an auxiliary graph with a dummy node r and edges of cost zero from r to any other node. In this graph we compute distances $d(v) := dist(r, v)$ from r in time $\tilde{O}(Mn^{\omega})$.

Given a query (s, t, e), we construct an auxiliary graph on node set $B \cup \{s, t\}$. For any pair $b_1, b_2 \in B$, we set the weight $w'(b_1b_2)$ of edge b_1b_2 to the minimum (precomputed) distance from b_1 to b_2 in any graph G_i not containing e. Since there are $O(\log n)$ such graphs, this step costs $\tilde{O}(|B|^2)$. At this point we set the distances from s to B and from B to t. It is here that our algorithm (for hop-long paths) deviates from [44]. In [44] the authors query the distance oracle for any pair (s, b) and (b, t) with $b \in B$. Since each query takes $\tilde{O}(n)$ time, altogether this costs $\tilde{O}(n|B|)$ time. We rather observe that, due to the lower bound part of Lemma 2.5, it is sufficient to consider only the shortest paths from s and to t which contain $\Omega(L)$ nodes. This costs only $\tilde{O}(n/L)$ by the final claim of Lemma 2.3. Therefore we are able to construct the auxiliary graph in $\tilde{O}(n/L \cdot |B| + |B|^2)$ time only. The rest of the query proceeds as in [44]: we add $d(u) - d(v)$ to each auxiliary weight $w'(uv)$, which makes edge weights non-negative. Then we use Dijkstra's algorithm to compute the shortest s-t path in the auxiliary graph in time $\tilde{O}(|B|^2)$. Summarizing, the preprocessing time for hop-long paths is $\tilde{O}(LMn^{\omega})$, and the query time is $\tilde{O}(n^2/L^2)$.

The overall failure probability is polynomially small in n by the usual arguments. Choosing $S = L^{\frac{1}{4-\omega}}$ completes the proof of Theorem 1.6.

6.4 Space Complexity

Consider first the DSO from Theorem 1.6. Note that for hop-average replacement paths it is sufficient to store, for each relevant distance oracle, only the portion corresponding to paths containing at least S nodes: this takes $O(n^2/S)$ space only. Altogether the space complexity is $\tilde{O}(n^2S + n^2L/S + (n/L)^2L) = \tilde{O}(n^2L^{\frac{1}{4-\omega}})$. For the DSO from Theorem 1.5, we need to add to the above space complexity a term $\tilde{O}(n^2|B|) = \tilde{O}(n^3/L)$. For a comparison, the DSO in [44] has space complexity $\tilde{O}(n^2L)$: this is always worse than $\tilde{O}(n^2L^{\frac{1}{4-\omega}})$, and worse than $\tilde{O}(n^3/L)$ for L large enough.

REFERENCES

[1] Alon, N., Galil, Z., and Margalit, O. 1997. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences 54,* 2, 255–262.

[2] Baswana, S. and Khanna, N. 2013. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica 66,* 1, 18–50.

[3] Bernstein, A. 2010. A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA).* 742–755.

[4] Bernstein, A. and Karger, D. R. 2008. Improved distance sensitivity oracles via random sampling. In *ACM-SIAM Symposium on Discrete Algorithms (SODA).* 34–43.

[5] Bernstein, A. and Karger, D. R. 2009. A nearly optimal oracle for avoiding failed vertices and edges. In *ACM Symposium on Theory of Computing (STOC)*. 101–110.
[6] Bhosle, A. M. and Gonzalez, T. F. 2004. Replacement paths for pairs of shortest path edges in directed graphs. In *International Conference on Parallel and Distributed Computing and Systems (PDCS)*. 94–99.
[7] Bilò, D., Grandoni, F., Gualà, L., Leucci, S., and Proietti, G. 2015. Improved purely additive fault-tolerant spanners. In *European Symposium on Algorithms (ESA)*. 167–178.
[8] Bodwin, G., Dinitz, M., Parter, M., and Vassilevska Williams, V. 2017. Optimal vertex fault tolerant spanners (for fixed stretch).
[9] Bodwin, G., Grandoni, F., Parter, M., and Williams, V. V. 2017. Preserving distances in very faulty graphs. *CoRR abs/1703.10293*. To appear in ICALP'17.
[10] Braunschvig, G., Chechik, S., Peleg, D., and Sealfon, A. 2015. Fault tolerant additive and (μ, α)-spanners. *Theoretical Computer Science 580*, 94–100.
[11] Byers, T. H. and Waterman, M. S. 1984. Determining all optimal and near optimal solutions when solving shortest path problems by dynamic programming. *Operations Research 32*, 1381–1384.
[12] Chechik, S., Cohen, S., Fiat, A., and Kaplan, H. 2017. $(1 + \epsilon)$-approximate f-sensitive distance oracles. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1479–1496.
[13] Chechik, S., Langberg, M., Peleg, D., and Roditty, L. 2009. Fault-tolerant spanners for general graphs. In *ACM Symposium on Theory of Computing (STOC)*. 435–444.
[14] Coppersmith, D. 1997. Rectangular matrix multiplication revisited. *Journal of Complexity 13,* 1, 42–49.
[15] Demetrescu, C., Thorup, M., Chowdhury, R. A., and Ramachandran, V. 2008. Oracles for distances avoiding a failed node or link. *SIAM Journal on Computing 37,* 5, 1299–1318.
[16] Dinitz, M. and Krauthgamer, R. 2011. Fault-tolerant spanners: better and simpler. In *ACM Symposium on Principles of Distributed Computing (PODC)*. 169–178.
[17] Duan, R. and Pettie, S. 2009. Dual-failure distance and connectivity oracles. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 506–515.
[18] Emek, Y., Peleg, D., and Roditty, L. 2010. A near-linear-time algorithm for computing replacement paths in planar directed graphs. *ACM Transactions on Algorithms 6,* 4.
[19] Eppstein, D. 1994. Finding the k shortest paths. In *IEEE Symposium on Foundations of Computer Science (FOCS)*. 154–165.
[20] Gall, F. L. 2012. Faster algorithms for rectangular matrix multiplication. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*. 514–523.
[21] Gall, F. L. 2014. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*. 296–303.
[22] Goldberg, A. V. 1995. Scaling algorithms for the shortest paths problem. *SIAM J. Comput. 24,* 3, 494–504.
[23] Gotthilf, Z. and Lewenstein, M. 2009. Improved algorithms for the k simple shortest paths and the replacement paths problems. *Information Processing Letters 109,* 7, 352–355.
[24] Grandoni, F. and Vassilevska Williams, V. 2012. Improved distance sensitivity oracles via fast single-source replacement paths. In *IEEE Symposium on Foundations of Computer Science (FOCS)*. 748–757.
[25] Hershberger, J. and Suri, S. 2001. Vickrey prices and shortest paths: what is an edge worth? In *IEEE Symposium on Foundations of Computer Science (FOCS)*. 252–259.
[26] Hershberger, J., Suri, S., and Bhosle, A. 2003. On the difficulty of some shortest path problems. In *Symposium on Theoretical Aspects of Computer Science (STACS)*. 343–354.
[27] Huang, X. and Pan, V. Y. 1998. Fast rectangular matrix multiplication and applications. *Journal of Complexity 14,* 2, 257–299.
[28] Johnson, D. B. 1977. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM 24,* 1, 1–13.
[29] Karger, D., Koller, D., and Phillips, S. 1993. Finding the hidden path: Time bounds for all-pairs shortest paths. *SIAM Journal on Computing 22,* 6, 1199–1217.
[30] Lawler, E. 1972. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science 18*, 401–405.
[31] Malik, K., Mittal, A. K., and Gupta, S. K. 1989. The k most vital arcs in the shortest path problem. *Operations Research Letters*, 223–227.
[32] Nardelli, E., Proietti, G., and Widmayer, P. 2001. A faster computation of the most vital edge of a shortest path. *Information Processing Letters 79,* 2, 81–85.
[33] Nisan, N. and Ronen, A. 2001. Algorithmic mechanism design. *Games and Economic Behavior 35*, 166–196.
[34] Parter, M. 2014. Vertex fault tolerant additive spanners. In *Symposium on Distributed Computing (DISC)*. 167–181.
[35] Parter, M. and Peleg, D. 2014. Fault tolerant approximate BFS structures. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1073–1092.

[36] Roditty, L. 2007. On the k-simple shortest paths problem in weighted directed graphs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 920–928.
[37] Roditty, L. and Zwick, U. 2005. Replacement paths and k simple shortest paths in unweighted directed graphs. In *International Colloquium on Automata, Languages and Programming (ICALP)*. 249–260.
[38] Seidel, R. 1995. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci. 51,* 3, 400–403.
[39] Shoshan, A. and Zwick, U. 1999. All pairs shortest paths in undirected graphs with integer weights. In *IEEE Symposium on Foundations of Computer Science (FOCS)*. 605–614.
[40] Thorup, M. 1999. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM 46,* 3, 362–394.
[41] Vassilevska Williams, V. 2011. Faster replacement paths. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1337–1346.
[42] Vassilevska Williams, V. 2012. Multiplying matrices faster than coppersmith winograd. In *ACM Symposium on Theory of Computing (STOC)*. 887–898.
[43] Vassilevska Williams, V. and Williams, R. 2010. Subcubic equivalences between path, matrix and triangle problems. In *IEEE Symposium on Foundations of Computer Science (FOCS)*. 645–654.
[44] Weimann, O. and Yuster, R. 2010. Replacement paths via fast matrix multiplication. In *IEEE Symposium on Foundations of Computer Science (FOCS)*. 655–662.
[45] Williams, R. 2014. Faster all-pairs shortest paths via circuit complexity. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*. 664–673.
[46] Yen, J. 1971. Finding the k shortest loopless paths in a network. *Management Science 17*, 712–716.
[47] Yuster, R. and Zwick, U. 2005. Answering distance queries in directed graphs using fast matrix multiplication. In *IEEE Symposium on Foundations of Computer Science (FOCS)*. 389–396.
[48] Yuval, G. 1976. An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications. *Information Processing Letters 4*, 155–156.
[49] Zwick, U. 2002. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM 49,* 3, 289–317.

Received ; revised ; accepted