Improved Distance Sensitivity Oracles via Fast Single-Source Replacement Paths

Fabrizio Grandoni
IDSIA, University of Lugano
fabrizio@idsia.ch

Virginia Vassilevska Williams
UC Berkeley and Stanford University
virgi@eecs.berkeley.edu

***Abstract*—A *distance sensitivity oracle* is a data structure which, given two nodes s and t in a directed edge-weighted graph G and an edge e, returns the shortest length of an s-t path not containing e, a so called *replacement path* for the triple (s,t,e). Such oracles are used to quickly recover from edge failures.**

In this paper we consider the case of integer weights in the interval $[-M,M]$, and 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 $\tilde{O}(n^{1-\alpha})$. Here $\omega < 2.373$ denotes the matrix multiplication exponent. For a comparison, 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].

The main novelty in our approach is an algorithm to compute all the replacement paths from a given source s, an interesting problem on its own. We can solve the latter *single-source replacement paths* problem in $\tilde{O}(APSP(n,M)))$ time, where $APSP(n,M) < \tilde{O}(M^{0.681}n^{2.575})$ [Zwick-JACM'02] is the runtime for computing all-pairs shortest paths in a graph with n vertices and integer edge weights in $[-M,M]$. For positive weights the runtime of our algorithm reduces to $\tilde{O}(Mn^{\omega})$. This matches the best known runtime for the simpler *replacement paths* problem in which both the source s and the target t are fixed [Vassilevska-SODA'11].

***Keywords*-replacement paths; distance sensitivity oracles; shortest paths.**

I. Introduction

Let $G=(V,E)$ be an n-node directed graph, with edge weights (or lengths) $w: E \to \mathbb{Q}$. 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. A *distance sensitivity oracle* (DSO) is a data structure that, after a preprocessing step, answers queries of the form (s,t,e) by returning the length $D_{s,t,e}$ of the corresponding replacement path $P_{s,t,e}$[1]. DSOs allow quick recovery from (single) edge failures.

DSOs are very well-studied in the literature. For arbitrary *non-negative* 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 every edge e in the shortest paths tree rooted at s, the new shortest paths tree from s in $G \setminus \{e\}$. (Here and throughout the paper, the $\tilde{O}$ notation suppresses $n^{o(1)}$ factors.) The queries are then answered in $O(1)$ time by looking up the answer. Similar DSOs can be obtained for graphs with possibly negative weights but no negative cycles by adding an extra $\tilde{O}(mn)$ time preprocessing step to replace all negative weights by non-negative ones, as in [12].

The preprocessing time for DSOs with arbitrary edge weights was improved to $\tilde{O}(mn^{\frac{3}{2}})$ by Demetrescu et al. [7], while keeping the query time constant. Bernstein and Karger further improved the preprocessing time to $\tilde{O}(\sqrt{m}n^2)$ [3] and finally to $\tilde{O}(mn)$ [4]. The latter preprocessing time matches, up to poly-logarithmic factors, the best known runtime for the *all-pairs shortest paths* problem (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. Let $\omega \in [2, 2.373]$ denote the smallest constant such that there is an algorithm to multiply two $n \times n$ matrices in $\tilde{O}(n^{\omega})$ time [21]. For integer weights in $[-M,M]$, Weimann and Yuster [22] 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[2]. An obvious open problem is whether one can achieve subcubic preprocessing time with

[1] In some variants of the problem the data structure also returns the replacement path itself. Our results can be modified to return paths as well, in time linear in the number of path edges.

[2] [22] 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.

linear (or even sublinear) query time. We answer this question affirmatively.

A. Our Results

Distance Sensitivity Oracles: In this paper we present an improved DSO in the case of integer weights of absolute value bounded by M.

Theorem 1. *For any integer $1 \leq S \leq n$, there is a randomized distance sensitivity oracle for 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[3] polynomially small in n.*

In particular, by choosing $S = \Theta(n^{\frac{1}{2(4-\omega)}})$, one obtains subcubic preprocessing time $\tilde{O}(Mn^{\omega+\frac{1}{2}}) < O(Mn^{2.88})$ and sublinear query time $\tilde{O}\left(n^{1-\frac{1}{2(4-\omega)}}\right) < O(n^{0.70})$. Our oracle is always better than the oracle in [22] in terms of query time, and for $\alpha < \frac{1}{2}$ it improves also on the preprocessing time.

Incidentally, we are also able to obtain a variant of the DSO in [22] with the same preprocessing time, but with an improved query time.

Theorem 2. *For any integer $1 \leq L \leq n$, there is a randomized distance sensitivity oracle for 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, we present it for the sake of completeness since it implies a strict improvement on [22] also for $\alpha > \frac{1}{2}$.

The DSO in [22] distinguishes between *hop-short* replacement paths, which contain at most $L = \Theta(n^{1-\alpha})$ nodes, and the remaining *hop-long* paths. Hop-short paths 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 [24] 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 take a completely different approach. Let B be a random sample of $\tilde{O}(n/L)$ nodes, so that whp 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 DSOs to Single Source Replacement Paths: From the above discussion, we can reduce the problem of designing an improved DSO to the problem of efficiently computing all the replacement path distances $D_{s,t,e}$ from a fixed source s ($s \in B$ in our DSO). We call this problem *single-source replacement paths* (SSRP). SSRP is a natural generalization of the well-studied *replacement paths* problem (RP) [2], [8], [9], [14], [15], [16], [20], [22], where both the source s and the target t are fixed. Somehow surprisingly, SSRP has not received much attention in the literature. The only reference to our knowledge is a paper by Hershberger et al. [10] that refers to the problem as *edge-replacement shortest paths trees* and shows that in the path-comparison model of computation of Karger et al. [13], SSRP on directed graphs with n nodes, m edges and arbitrary edge weights requires $\Omega(mn)$ comparisons. The reductions by Vassilevska Williams and Williams [23] imply that, for arbitrary weights, any $\tilde{O}(n^{3-\varepsilon})$ algorithm for a constant $\varepsilon > 0$ even for the simpler RP problem would imply a $\tilde{O}(n^{3-\delta})$ algorithm for some constant $\delta > 0$ for all-pairs shortest paths (APSP). Therefore, there seems to be little hope to obtain a subcubic time algorithm for SSRP. Here we present the first subcubic algorithms for small integer weights. We note that due to our use of fast matrix multiplication, our algorithms do not fall under the path-comparison model. We think that SSRP is a natural problem in itself, regardless of its application to DSOs.

Theorem 3. *There is a randomized algorithm that solves SSRP in 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.*

Note that the runtime of our SSRP algorithm for weights in $[-M, M]$ matches the runtime of Zwick's APSP algorithm[4] [26]. We also remark that the runtime

[3] 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.

[4] Zwick achieves a slightly faster runtime $O(M^{0.681} n^{2.575})$ by means of fast rectangular matrix multiplication [11]. We can similarly improve the runtime of our algorithm: we will omit these technical details for the sake of presentation.

of our algorithm for positive weights matches the best known algorithm by Vassilevska Williams [20] 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 for 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 $D_{v,u}$ 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 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 (wlog) 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 $v = s$ 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 the algorithm in [20].

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 [16] 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 whp 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 [24] 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 4. *There is a randomized algorithm that solves STSP in 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.*

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 bound on ω of 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 $[X, 2X)$, $X \geq L$, and search for the detours with a number of nodes in the interval by considering the detours which 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. One possibility is to compute 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. However, in order to design a fast DSO, it is convenient to exploit a variant of this 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 logarithmic set of intervals $[X, 2X)$, $X \geq L$, we address the problem of computing replacement paths with a number of nodes in the 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 which start from the *first* $2X$ nodes of P' (otherwise the corresponding replacement paths would have $> 2X$ nodes). The runtime of the modified SSRP 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., $\tilde{O}(n/X)$). This is sufficient to 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).

B. Related Work

The replacement paths problem (RP) is very well studied in the literature. Let m denote the number of edges and n the number of vertices in the following. Malik, Mittal and Gupta [14] gave an $\tilde{O}(m)$ time algorithm for RP in undirected weighted graphs. Nardelli et al. [15] used Thorup's linear time algorithm for single source shortest paths [19] to improve the runtime to $O(m\alpha(n))$ in the word-RAM model of computation. The problem in directed graphs seems somewhat more difficult. Roditty and Zwick [16] showed that in unweighted directed graphs RP can be solved in $\tilde{O}(m\sqrt{n})$ time. For weighted planar digraphs, the runtime can be reduced to $\tilde{O}(n)$ as shown by Emek, Peleg and Roditty [8]. For arbitrary directed graphs with arbitrary edge weights, the fastest known algorithm for RP is by Gotthilf and Lewenstein [9] and runs in $O(mn + n^2 \log\log n)$ time. For dense graphs with arbitrary edge weights, nothing much better than cubic time is known.[5] Vassilevska Williams and Williams [23] showed that the RP problem in directed graphs is equivalent to APSP, under subcubic reductions, i.e. essentially either both problems admit truly subcubic algorithms, or neither of them does. This apparent cubic time barrier only holds when one wants to solve the RP problem *exactly*. In contrast, Bernstein [2] described an algorithm for RP in directed graphs with positive weights that for any $\varepsilon > 0$, computes $(1+\varepsilon)$-approximate replacement paths in $\tilde{O}(\frac{1}{\varepsilon}m)$ time.

For graphs with bounded integer weights, there are two improved algorithms for RP based on fast matrix multiplication. Let $\omega \in [2, 3]$ denote the smallest constant such that for all $\varepsilon > 0$, there is an algorithm to multiply two $n \times n$ matrices in $O(n^{\omega+\varepsilon})$ time. The best bound on ω was 2.376 for about twenty years [6], and it was recently improved independently by Stothers [18] to 2.374 and by Vassilevska Williams [21] to 2.373. For directed graphs with integer weights in $[-M, M]$, Weimann and Yuster [22] described an algorithm for RP of runtime $\tilde{O}(Mn^{1+\frac{2}{3}\omega})$. Vassilevska Williams [20] designed a faster randomized algorithm with runtime $\tilde{O}(Mn^{\omega})$, and failure probability polynomially small in n. We will use her algorithm as a black box. An RP algorithm can be used also to compute the k shortest paths from s to t with an increase of the running time by a factor $O(k)$ [16].

Matrix multiplication is a common tool to solve shortest path problems in the presence of small integer weights. Alon, Galil and Margalit [1] presented an algorithm for APSP in directed graphs with weights in $\{-1, 0, 1\}$ of runtime $\tilde{O}(n^{\frac{\omega+3}{2}})$. This was improved and generalized to integer weights in $[-M, M]$ by Zwick [26], whose algorithm has runtime $\tilde{O}(M^{\frac{1}{4-\omega}}n^{2+\frac{1}{4-\omega}})$ (or $O(M^{0.681}n^{2.575})$ by using fast rectangular matrix multiplication). For undirected graphs, the current best runtime is $\tilde{O}(Mn^{\omega})$ by Shoshan and Zwick [17]. Yuster and Zwick [24] gave the currently fastest $\tilde{O}(Mn^{\omega})$ algorithm for the single-source shortest path problem (SSSP) in directed graphs with integer weights in $[-M, M]$.

C. Preliminaries

The distance product $A \star B$ of an $a \times b$ matrix A and 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 [25], show:

Lemma 1. *The distance product of an $a \times b$ matrix by a $b \times c$ matrix, where each entry is either an integer in $[-M, M]$ or $+\infty$, can be computed in time $\tilde{O}(\min\{abc, M \cdot \frac{abc}{(\min\{a,b,c\})^{3-\omega}}\})$.*

The following lemma is at the heart of the DSO in [22], and it will be crucial for us as well.

Lemma 2. *[22] 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 whp: (a) For any edge $e \in G$, there are $\Theta(\log n)$ graphs G_i not containing e. (b) For any*

[5] Polylogarithmic improvements are possible. For example, Chan's [5] $O(n^3 \log\log^3 n/\log^2 n)$ algorithm for all-pairs shortest paths can be converted to one for replacement paths.

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 $P_{s,t,e}$.

We will exploit the distance oracle in [24]. The final claim of the next lemma is implicit in [24].

Lemma 3. *[24] Given a directed graph with integer weights in $[-M, M]$, there exists a distance oracle with preprocessing time $\tilde{O}(Mn^\omega)$, which answers queries (s,t) by returning the length of the shortest s-t path in $O(n)$ time. Furthermore, the distance between nodes with a shortest path containing at least L nodes can be returned in $\tilde{O}(n/L)$ time.*

We will extensively use the following lemma. While the upper bound part is used also in [22], the lower bound part is at the base of the improved variant in Theorem 2.

Lemma 4. *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. 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 $L/8$ and at most L nodes each.*

Proof: We show that the claim does not hold for a fixed path P between nodes s to t, with probability at most ne^{-N}: the claim follows from the union bound. If P contains at most L nodes the claim is trivially true. Otherwise, let us iteratively remove the first $L/2$ nodes of P until it contains less than L nodes. This partitions P into at most $\frac{2n}{L}$ intervals containing $L/2$ nodes and a last interval which might contain up to (almost) L nodes but also contains at least $L/2$ nodes. Let us consider the central subinterval of $L/4$ nodes of each interval, except possibly the last one. If the last interval has more than $L/2$ nodes, then consider its first subinterval of $L/4$ nodes.

Suppose that each 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 $\frac{2n}{L}(1 - \frac{L}{4n})^{\frac{4Nn}{L}} \leq ne^{-N}$. ∎

Throughout this paper $C > 0$ denotes a sufficiently large constant that guarantees that the failure probability of the considered algorithms is at most n^{-Q} for any fixed constant $Q > 0$.

II. Solving STSP

Our STSP algorithm builds upon Zwick's APSP algorithm [26], by combining a new idea with an approach from [24]. Zwick's algorithm proceeds as follows. 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 which 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 whp $D[u,v] = dist(u,v)$. By Lemma 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 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 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 $|B_i| = p_i \cdot |V|$.
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 [24] 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 [24] directly.

Proof: (Theorem 4) The correctness analysis follows along the same line as in [26] and [24]. Consider next the runtime. Let us assume $\sigma := |S| \leq |T| =: \tau$, the other case is 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 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. ∎

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

III. A Faster SSRP Algorithm

We show how to solve SSRP, by reducing it to a small (subpolynomial) set of subpath problems and a small number of recursive calls on much smaller instances (Section III-A), and by solving each subpath problem efficiently (Section III-B).

A. From SSRP to Subpath Problems

Consider an SSRP instance on n nodes with weights of absolute value at most M. For technical reasons we introduce a parameter $\tilde{n} \geq n$ which is implicitly passed in all the recursive calls. Intuitively, $\tilde{n}$ is the number of nodes in the root of the recursion tree (i.e., in the input instance). Let $H := h(\tilde{n})$ be a suitably chosen subpolynomial function[6].

Assume we are given the shortest paths tree T_s of the current instance and the lengths $D_{v,u}$ of its paths $P_{v,u}$ (this can be computed in $\tilde{O}(Mn^\omega)$ time). We define a matrix $\{\tilde{D}_{s,t,e}\}_{(t,e)\in V(T_s)\times E(T_s)}$ which contains our estimates of the replacement path distances $D_{s,t,e}$ for any node t and edge e in T_s. Observe that for $e \notin E(T_s)$, $D_{s,t,e} = D_{s,t}$ and this information can be stored implicitly.

Let us assume that $n \geq H$ and $M \leq n^{3-\omega}$, otherwise we solve the problem with the trivial cubic algorithm in $\tilde{O}(n^3) = \tilde{O}(Mn^\omega)$ time. The algorithm initializes each $\tilde{D}_{s,t,e}$ to $+\infty$, and then sets $\tilde{D}_{s,t,e} \leftarrow D_{s,t}$ for any pair (t,e) such that $e \notin E(P_{s,t})$.

[6]For $\omega > 2$, one can set $H = \operatorname{poly}\log(\tilde{n})$ and otherwise $H = 2^{\sqrt{\log\tilde{n}\log\log\tilde{n}}}$.

Then the algorithm partitions T_s into $\Theta(H)$ subtrees with $\Theta(n/H)$ nodes each. In more detail, starting from $\mathcal{T} = \{T_s\}$, while there is $T \in \mathcal{T}$ with more than n/H nodes, we compute a *balanced separator* node v of T, and use it to split T into two trees T' and T'', with $|V(T')|, |V(T'')| \geq \frac{1}{3}|V(T)| + 1$, which partition the edges of T and intersect at node v only[7]. The trees T' and T'' replace T in $\mathcal{T}$. Observe that at the end of the process, the trees in $\mathcal{T}$ are edge-disjoint though they might intersect at separator nodes. Furthermore, these trees partition the edge set of T_s. As each tree contains at least $\frac{n}{3H}$ edges, $|\mathcal{T}| \leq 3H$. Each splitting step, and hence the overall procedure, can be performed in $\tilde{O}(n)$ time.

Let P' be the path from s to the root t' of $T' \in \mathcal{T}$. The values of $\tilde{D}_{s,t,e}$ that still need to be updated must satisfy either (a) $(t,e) \in V(T') \times E(P')$ or (b) $(t,e) \in V(T') \times E(T')$ for some $T' \in \mathcal{T}$. We define the subproblem of computing $D_{s,t,e}$ for the pairs (t,e) of the first and second type a *subpath problem* (P', T') and a *subtree problem* T', respectively.

In the next section we show how to solve a subpath problem with failure probability polynomially small in n. We apply that procedure for $O(\log\tilde{n})$ times to each subpath problem, so that the failure probability becomes polynomially small in $\tilde{n}$.

We solve each subtree problem T' recursively, after a proper *compression* step where we reduce the number of nodes to $O(\frac{n\log\tilde{n}}{H})$. ($T'$ is already small, however, there are many nodes outside of T', and the compression step aims at reducing these.) In more detail, we sample a random set B' of $\frac{n}{H}\cdot C\log\tilde{n}$ nodes as in Lemma 4, so that with probability at least $1 - 1/\operatorname{poly}(\tilde{n})$ each replacement path on at least H nodes is partitioned by B' into segments containing at most H nodes each. We construct an auxiliary directed complete graph G' on node set $B' \cup V(T') \cup \{s\}$, whose edges e are labelled with the corresponding shortest path distance $w'(e)$ in the graph $G - E(T')$ but only considering paths on at most H nodes, hence of absolute length at most MH. Running the truncated version of Zwick's APSP algorithm from Corollary 1 $O(\log\tilde{n})$ times, the computation of the weights $w'(\cdot)$ takes $\tilde{O}(Mn^\omega H^{3-\omega}\log\tilde{n})$ time and succeeds with probability at least $1 - 1/poly(\tilde{n})$. Then we add back edges $e \in E(T')$ with their original weight $w'(e) = w(e)$. Observe that the multi-graph G' contains a *contracted* representative of each replacement path for the considered triples (s,t,e) with failiure probability polynomi-

[7]We recall that in $O(n)$ time we can compute a balanced separator node v and a partition of $T - \{v\}$ in two forests F' and F'', so that each forest contains at least $\frac{1}{3}|V(T)|$ nodes. It is then sufficient to let T' and T'' be the subtrees induces by F' and F'', respectively, plus node v.

ally small in $\tilde{n}$.

We get rid of parallel edges as follows: let $e = uv \in E(T')$ and $f = uv$ be the edge parallel to e (and with the same orientation). Observe that any replacement path containing the subpath corresponding to f must be a replacement path for edge e and only for that edge. We set $\tilde{D}_{s,t,e} \leftarrow \min\{\tilde{D}_{s,t,e}, D_{s,t} - w(e) + w'(f)\}$ for each t such that (t,e) is relevant, and remove f from G'.

Observe that the shortest paths tree T'_s from s in G' contains the subtree T' (by breaking ties properly). We then solve the SSRP problem induced by (s, G', w') recursively. Let $\tilde{D}^{subtree}_{s,t,e}$ be the obtained lengths. Finally, for each $(t,e) \in V(T') \times E(T')$, we set $\tilde{D}_{s,t,e} \leftarrow \min\{\tilde{D}_{s,t,e}, \tilde{D}^{subtree}_{s,t,e}\}$.

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

Lemma 5. *Given an algorithm that solves a subpath problem in time $\tilde{O}(M^\alpha n^\beta)$ with failure probability $1-\delta$, for constants $\delta > 0$ 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 polynomially small in n.*

Proof: Let n and M denote the number of nodes and maximum absolute weight in the input SSRP instance. We will use $\overline{n}$ and $\overline{M}$ for the same quantities in some SSRP subproblem. We let $H = h(n) = 2^{\sqrt{\log n \log\log n}}$ (assuming $\omega > 2$, $h(n) = \text{poly}\log(n)$ would be sufficient).

We analyze the initial call of the above algorithm where the role of $\tilde{n}$ is played by n. 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. By solving each subpath problem for $C \log n$ times and storing the best distances, we obtain a subpath algorithm with failure probability n^{-Q} for any given constant $Q > 0$. We assume that this is the subpath procedure used in the considered algorithm. As already discussed, the compression step in each subtree problem preserves the correct replacement path distances with a similar failure probability n^{-Q}, since both the sampling step and the computation of shortest paths fail with polynomially small probability in n. The claim on the failure probability of the entire algorithm follows from the union bound.

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

Consider first the compression steps. Each compression step can be performed in time $\overline{M}\overline{n}^\omega g(n)$ 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

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

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

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

The same bound holds, with a factor $3H$ less, for the base SSRP instances with $\overline{M} > \overline{n}^{3-\omega}$ which are solved in $\tilde{O}(\overline{n}^3) \leq \tilde{O}(\overline{M}\overline{n}^\omega)$ time each.

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

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

and hence their total execution time is at most

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

The claim on the running time follows. ∎

B. Solving Subpath Problems

Let us focus on a subpath problem (P', T'), where s and t' are the endpoints of P'. Let n and M be the number of nodes and the largest absolute weight in the considered instance, respectively. In the worst case both P' and T' contain $O(n)$ nodes. As mentioned in the introduction, we distinguish between two types of replacement paths $P_{s,t,e}$ for $(t,e) \in V(T') \times E(P')$, $e = uv$. A *jumping path* $P_{s,t,e}$ leaves P' at some node (between s and u) and then meets P' again at some other node (between v and t'). A *departing path* $P_{s,t,e}$ leaves P' at some node (between s and u) and never meets P' again.

We can easily deal with jumping paths via a reduction to the RP problem: we solve the RP problem induced by P' with the $\tilde{O}(Mn^\omega)$ time algorithm in [20]. Let $\tilde{D}_{s,t',e}$ be the resulting distances, which are incorrect with polynomially small probability in n. Then the shortest jumping path length for the triple (s,t,e) is simply $\tilde{D}_{s,t',e} + D_{t',t}$: this takes $\tilde{O}(n^2)$ extra time.

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

Lemma 6. *There is an algorithm which solves a given subpath problem with integer weights in $[-M,M]$ in time $\tilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$, with failure probability polynomially small in n.*

The part of Theorem 3 relative to negative weights follows from Lemmas 5 and 6.

The rest of this section is devoted to positive weights. Let V_x be the final x nodes of 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_{ML}, since otherwise a detour would be shorter than the shortest path between its endpoints or the associated replacement path would be jumping. We use the STSP algorithm from Theorem 4 to compute the distances $dist_{G'}(v,t)$ from $v \in V_{ML}$ to all $t \in V(T')$ in G'. For the remaining detours, let us define $O(\log n)$ intervals $[X_i, 2X_i)$ with $X_i = 2^i L$ and $0 \le i \le \lceil \log_2 \frac{n}{L} \rceil$. 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 $\frac{n}{X_i} \cdot C \log n$ nodes as in Lemma 4, so that B_i hits any detour on at least X_i nodes whp. We compute $dist_{G'}(v,b)$ and $dist_{G'}(b,v)$ for any $b \in B_i$ and any node v, using our STSP algorithm. For each $(v,t) \in V_{2MX_i} \times V(T_i)$, the desired distance is $\min_{b \in B_i}\{dist_{G'}(v,b) + dist_{G'}(b,t)\}$.

Lemma 7. *There is an algorithm which solves a given subpath problem with integer weights in $[1,M]$ in time $\tilde{O}(Mn^\omega)$, 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 $\tilde{O}(Mn^\omega)$ time. The computation of the detours themselves takes time

$$\begin{aligned}&\tilde{O}(Mn^\omega + ML\, n\, (Mn)^{\frac{1}{4-\omega}}) \\ &+ \sum_i \tilde{O}(Mn^\omega + \frac{n}{X_i} n (Mn)^{\frac{1}{4-\omega}} + MX_i \frac{n}{X_i} n) \\ &= \tilde{O}(Mn^\omega + (MLn + \frac{n^2}{L})(Mn)^{\frac{1}{4-\omega}}).\end{aligned}$$

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

Combining Lemmas 5 and 7, one obtains the part of Theorem 3 corresponding to positive weights (note that the compression step in Lemma 5 does not create negative weights).

IV. Distance Sensitivity Oracles

In this section we present our improved DSOs. We start with the DSO from Theorem 1: we consider first the case of positive integer weights (Section IV-A), and later extend the result to allow for non-positive weights (Section IV-B). In Section IV-C we describe the alternative DSO from Theorem 2. We conclude the section with a brief discussion about the space complexity.

A. Positive Weights

The basic strategy is as follows. Given two integer parameters $0 \le S \le L \le n$, 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 $G_1, \dots, G_{S \cdot C \log n}$ as in Lemma 2. We compute all-pairs shortest paths on at most S nodes in each G_i as in Corollary 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 whp 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. We apply the preprocessing step of the distance oracle from Lemma 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 and 3, whp 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 4, so that whp 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 whp 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 for positive weights.

B. 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 which 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, this also upper-bounds the overall runtime of the algorithm.

Lemma 8. *For any $0 \le X \le n$, 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 $L \le X_i := 2^i L < 2n$. 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 4, so that whp 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 $\tilde{O}(\frac{n}{X_i} Mn^\omega + \frac{n}{X_i} X_i M^{\frac{1}{4-\omega}} n^{1+\frac{1}{4-\omega}}) \le \tilde{O}(\frac{n}{L} Mn^\omega + M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$. For a query (s,t,e), it is sufficient to consider all the triples (s,b,t) with $b \in B_i$, which takes $\tilde{O}(\frac{n}{X_i}) \le \tilde{O}(\frac{n}{L})$ time. Since $LMn^\omega + \frac{n}{L} Mn^\omega \ge Mn^{\omega+\frac{1}{2}} \ge M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}}$ 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. 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.

C. An Alternative Oracle

We next describe the DSO from Theorem 2. We again distinguish between hop-short, hop-average, and hop-long paths. We handle the first two types of paths as in Section IV. 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 3 for each such graph. We sample $\frac{n}{L} \cdot C \log n$ nodes B as in Lemma 4, 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 [22]. 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 $w'(b_1 b_2)$ of edge $b_1 b_2$ 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 [22]. In [22] 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 4, 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 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 [22]: 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 2.

D. Space Complexity

Consider first the DSO from Theorem 2. 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, 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 [22] 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.

ACKNOWLEDGMENT

This work was partially supported by ERC Starting Grant NEWNET 279352, by NSF Grants CCF-0830797, CCF-1118083, IIS-0963478 and IIS-0904325, and by AFOSR MURI Grant.

REFERENCES

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

[2] A. Bernstein. A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In *Proc. SODA*, pages 742–755, 2010.

[3] A. Bernstein and D. R. Karger. Improved distance sensitivity oracles via random sampling. In *Proc. SODA*, pages 34–43, 2008.

[4] A. Bernstein and D. R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *Proc. STOC*, pages 101–110, 2009.

[5] T. M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In *Proc. STOC*, pages 590–598, 2007.

[6] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9(3):251–280, 1990.

[7] C. Demetrescu, M. Thorup, R. A. Chowdhury, and V. Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM Journal on Computing*, 37(5):1299–1318, 2008.

[8] Y. Emek, D. Peleg, and L. Roditty. A near-linear-time algorithm for computing replacement paths in planar directed graphs. *ACM Transactions on Algorithms*, 6(4), 2010.

[9] Z. Gotthilf and M. Lewenstein. Improved algorithms for the k simple shortest paths and the replacement paths problems. *Information Processing Letters*, 109(7):352–355, 2009.

[10] J. Hershberger, S. Suri, and A. Bhosle. On the difficulty of some shortest path problems. In *Proc. STACS*, pages 343–354, 2003.

[11] X. Huang and V. Y. Pan. Fast rectangular matrix multiplication and applications. *Journal of Complexity*, 14(2):257–299, 1998.

[12] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977.

[13] D. Karger, D. Koller, and S. Phillips. Finding the hidden path: Time bounds for all-pairs shortest paths. *SIAM Journal on Computing*, 22(6):1199–1217, 1993.

[14] K. Malik, A. K. Mittal, and S. K. Gupta. The k most vital arcs in the shortest path problem. *Operations Research Letters*, pages 223–227, 1989.

[15] E. Nardelli, G. Proietti, and P. Widmayer. A faster computation of the most vital edge of a shortest path. *Information Processing Letters*, 79(2):81–85, 2001.

[16] L. Roditty and U. Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. In *Proc. ICALP*, pages 249–260, 2005.

[17] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proc. FOCS*, pages 605–614, 1999.

[18] A. Stothers. *Ph.D. Thesis, U. Edinburgh*, 2010.

[19] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46(3):362–394, 1999.

[20] V. Vassilevska Williams. Faster replacement paths. In *Proc. SODA*, pages 1337–1346, 2011.

[21] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith Winograd. In *Proc. STOC*, pages 887-898, 2012.

[22] O. Weimann and R. Yuster. Replacement paths via fast matrix multiplication. In *Proc. FOCS*, pages 655–662, 2010.

[23] V. Vassilevska Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proc. FOCS*, pages 645–654, 2010.

[24] R. Yuster and U. Zwick. Answering distance queries in directed graphs using fast matrix multiplication. In *Proc. FOCS*, pages 389–396, 2005.

[25] G. Yuval. An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications. *Information Processing Letters*, 4:155–156, 1976.

[26] U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002.