Replacement Paths and Distance Sensitivity Oracles via Fast Matrix Multiplication*

OREN WEIMANN, University of Haifa
RAPHAEL YUSTER, University of Haifa

A *distance sensitivity oracle* of an n-vertex graph $G = (V, E)$ is a data structure that can report shortest paths when edges of the graph fail. A query $(u \in V, v \in V, S \subseteq E)$ to this oracle returns a shortest u-to-v path in the graph $G' = (V, E \setminus S)$. We present randomized (Monte Carlo) algorithms for constructing a distance sensitivity oracle of size $\tilde{O}(n^{3-\alpha})$ for $|S| = O(\lg n / \lg \lg n)$ and any choice of $0 < \alpha < 1$. For real edge-lengths, the oracle is constructed in $O(n^{4-\alpha})$ time and a query to this oracle[1] takes $\tilde{O}(n^{2-2(1-\alpha)/|S|})$ time. For integral edge-lengths in $\{-M, \ldots, M\}$, using the current $\omega < 2.376$ matrix multiplication exponent, the oracle is constructed in $O(Mn^{3.376-\alpha})$ time with $\tilde{O}(n^{2-(1-\alpha)/|S|})$ query, or alternatively in $O(M^{0.681}n^{3.575-\alpha})$ time with $\tilde{O}(n^{2-2(1-\alpha)/|S|})$ query.

Distance sensitivity oracles generalize the *replacement paths* problem in which u and v are known in advance and $|S| = 1$. In other words, if P is a shortest path from u to v in G, then the replacement paths problem asks to compute, for every edge e on P, a shortest u-to-v path that avoids e. Our new technique for constructing distance sensitivity oracles using fast matrix multiplication also yields the first sub-cubic time algorithm for the replacement paths problem when the edge-lengths are small integers. In particular, it yields a randomized (Monte Carlo) $\tilde{O}(Mn^{2.376} + M^{\frac{2}{3}}n^{2.584})$ time algorithm for the replacement paths problem assuming $M \leq n^{0.624}$.

Finally, we mention that both our replacement paths algorithm and our distance sensitivity oracle can be made to work, in the same time and space bounds, for the case of failed vertices rather than edges, That is, when S is a set of vertices and we seek a shortest u-to-v path in the graph obtained from G by removing all vertices in S and their adjacent edges.

Categories and Subject Descriptors: G.2.2 [**Graph Theory**]: Path and circuit problems, Graph algorithms; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems – *Computations on discrete structures*; E.1 [**Data Structures**]: Graphs and networks

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Shortest paths, replacement paths, distance oracles, fault tolerance

ACM Reference Format:
Weimann, O. and Yuster, R. 2012. Replacement Paths and Distance Sensitivity Oracles via Fast Matrix Multiplication. *ACM Trans. Algor.* 0, 0, Article 0 (2012), 13 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

Author's addresses: O. Weimann, Computer Science Department, University of Haifa, Haifa, Israel; R. Yuster, Mathematics Department, University of Haifa, Haifa, Israel; oren@cs.haifa.ac.il, raphy@math.haifa.ac.il

* A preliminary version of this paper appeared in FOCS 2010 [Weimann and Yuster 2010]

DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Resilience to failures is an important requirement of modern networks. In a failure event, some of the network's vertices and edges may be temporarily unavailable. As a consequence, structural information of the network such as its connectivity and distance metric might change and should be updated efficiently. In this paper we focus on maintaining distances in a directed weighted graph where one or more vertices or edges are unavailable due to failure. Specifically, we would like to answer queries of the form (u, v, S) seeking a shortest u-to-v path that avoids all the edges (or vertices) in S. A data structure that can support such queries is called a *distance sensitivity oracle*.

Existing Oracles. For the case of a single edge or vertex fault (when $|S| = 1$), Demetrescu et al. [Demetrescu et al. 2008] presented a distance sensitivity oracle that can be constructed in $\tilde{O}(mn^2)$ time and $O(n^2 \lg n)$ space and has constant query time. Bernstein and Karger improved the preprocessing to $O(n^2\sqrt{m})$ [Bernstein and Karger 2008] and then further to $\tilde{O}(mn)$ [Bernstein and Karger 2009], with unchanged constant query time. For the case of two failures (two vertices or two edges), Duan and Pettie [Duan and Pettie 2009] construct an oracle in polynomial time and $\tilde{O}(n^2)$ space, whose query time is $O(\lg n)$.

For more than two edge-failures, Chechik et al. [Chechik et al. 2010] showed that if we are willing to settle for approximate distances, then for any integer $k > 1$, an oracle of size $O(|S|kn^{1+1/k} \lg(nM))$ (where M is the heaviest edge-length in the graph) constructed in polynomial time can handle $|S|$ *edge failures* in $\tilde{O}(|S|)$ query time. Their oracle returns distances with stretch (approximation ratio) $(8k - 2)(f + 1)$, and only works for edge-faults. For exact distances however, the only known solution is the naive one: upon query (u, v, S) remove all edges of S and run a shortest paths algorithm (e.g., Dijkstra) from u to v in $O(n^2)$ time. Our goal in this paper is thus to construct an oracle that answers queries in sub-quadratic time.

We also mention the related problem of maintaining distance queries while updating the graph (deleting and inserting edges and vertices). Demetrescu and Italiano [Demetrescu and Italiano 2004] presented a data structure to maintain distance queries in constant time with $\tilde{O}(n^2)$ amortized update. Their algorithm was slightly improved by Thorup [Thorup 2004]. For *unweighted undirected* graphs, Roditty and Zwick [Roditty and Zwick 2004] obtained a $(1 + \varepsilon)$-approximation algorithm with an expected amortized update time of $\tilde{O}(mn/t)$ and worst-case query time of $O(t)$. Thorup [Thorup 2005] obtained $O(n^{2.75})$ *worst case* updates.

Replacement Paths. The replacement paths problem is an important restricted version of distance sensitivity oracles with one edge fault, where we know the vertices u and v in advance. Given vertices u and v, let P be a shortest u-to-v path in the original graph G. The replacement paths problem asks to compute, for every edge e in P, a shortest u-to-v path that avoids e. This generalization of the fundamental *shortest paths* problem is strongly motivated by two applications. In auction theory, the replacement paths problem is used to compute the Vickrey pricing of edges owned by selfish agents [Nisan and Ronen 2001; Hershberger and Suri 2001]. Another application is that of computing the k shortest simple paths between a pair of vertices. This problem reduces to running k executions of a replacement paths algorithm, and has many applications itself [Eppstein 1999].

The naive solution to the replacement paths problem is to remove each edge e on P, one at a time, and compute a shortest s-to-t path each time. This can be done in $O(mn + n^2 \lg n)$ time. Except for the slight improvement of Gotthilf and Lewenstein [Gotthilf and Lewenstein 2009] to $O(mn + n^2 \lg \lg n)$, no faster algorithms are known for weighted, directed graphs. Similarly, the fastest algorithm for finding k

shortest simple paths, given by Yen [Yen 1971] and Lawler [Lawler 1972], uses k replacement path computations and therefore runs in $O(k(mn + n^2 \lg\lg n))$ time. Hershberger et al. [Hershberger et al. 2003] gave an $\Omega(m\sqrt{n})$ lower bound for both these problems in the path-comparison model of Karger et al. [Karger et al. 1993].

Overcoming the $o(mn)$ bound for the replacement paths problem (and the $o(kmn)$ bound for k shortest simple paths) has received a lot of attention recently. There were two directions taken. The first was to consider special graph classes. For *undirected* graphs, the problem is significantly easier. Malik et al. [Malik et al. 1989] presented an $\tilde{O}(m)$ time algorithm for the replacement paths problem, that was later extended to avoid vertices rather than edges [Nardelli et al. 2003]. Nardelli et al. [Nardelli et al. 2001] gave an $O(m\alpha(m, n))$ time algorithm for the problem, in a stronger model of computation, using the linear time single source shortest paths algorithm of Thorup [Thorup 1999]. For directed *unweighted* graphs, Roditty and Zwick [Roditty and Zwick 2005] gave an $\widetilde{O}(m\sqrt{n})$ time randomized algorithm. Finally, for the class of directed *planar* graphs, Emek et al. [Emek et al. 2008] gave an $O(n \lg^3 n)$ time algorithm, improved in [Klein et al. 2009] and later in [Wulff-Nilsen 2010] to $O(n \lg n)$.

The second way of overcoming the $o(mn)$ bound was to settle for *approximate* distances. Roditty [Roditty 2007] showed that approximation can in fact beat the $O(kmn)$ bound for k shortest simple paths. In particular, Roditty presented an $\tilde{O}(km\sqrt{n})$-time $3/2$-approximation algorithm for finding the k shortest simple s-to-t paths in directed graphs with positive edge lengths. Bernstein [Bernstein 2010] improved this to a $(1+\varepsilon)$-approximation algorithm requiring $\tilde{O}(km/\varepsilon)$ time. Bernstein also gave the first approximation algorithm for the replacement paths problem. His replacement paths algorithm is a $(1+\varepsilon)$-approximation running in $\tilde{O}(m \lg(nC/c)/\varepsilon)$ time, where C is the largest edge length in the graph and c is the smallest.

Our Results. Our main result is a randomized algorithm for constructing a sub-cubic space distance sensitivity oracle that answers queries (u, v, S) with $|S| = o(\lg n/\lg\lg n)$ in sub-quadratic (notably, sub-Dijkstra) time. The exact bounds are stated by the following theorem.

THEOREM 1.1. *For any $0 < \alpha < 1$ and any $1 \le f \le (1-\alpha)\lg n/\lg\lg n$, there is a distance sensitivity oracle of size $\tilde{O}(n^{3-\alpha})$ for avoiding a set S (of either edges or vertices) with $|S| \le f$ on an n-vertex directed graph. For real edge-lengths, the oracle is constructed in $O(n^{4-\alpha})$ time and a query takes $\tilde{O}(n^{2-2(1-\alpha)/f})$ time. For integral edge-lengths in $\{-M, \ldots, M\}$, the oracle is constructed in $O(Mn^{3.376-\alpha})$ time with $\tilde{O}(n^{2-(1-\alpha)/f})$ query time, or alternatively in $O(M^{0.681}n^{3.575-\alpha})$ time with $\tilde{O}(n^{2-2(1-\alpha)/f})$ query time.*

As a special case of our distance sensitivity oracle, we obtain a randomized algorithm for solving the replacement paths problem in the following time bounds.

COROLLARY 1.2. *The replacement paths problem can be solved in $\tilde{O}(Mn^{\omega} + M^{\frac{2}{3}}n^{1+\frac{2}{3}\omega})$ time on an n-vertex directed graph with integral edge-lengths in $\{-M, \ldots, M\}$.*

This algorithm is the first sub-cubic time algorithm for the replacement paths problem for weighted graphs and the first to use fast matrix multiplication. Since the FOCS'10 conference version of this paper [Weimann and Yuster 2010], Vassilevska-Williams in SODA'11 [Vassilevska-Williams 2011] has managed to reduce the time complexity of the replacement paths problem to $\tilde{O}(Mn^{\omega})$. Her solution uses some of the ideas that we present here as well as some new and elegant ideas. However, it is

not known how to extend her solution to a distance sensitivity oracle (even not when $|S| = 1$).

Finally, we mention that both our replacement paths algorithm and our distance sensitivity oracle can be made to work, in the same time bounds, for the case of *failed vertices* rather than edges. To make the presentation simpler, the paper focuses on the case of failed edges. In Section 5 we explain why the case of failed vertices can be handled similarly (in the same bounds of Theorem 1.1).

Technique. There are two extremes for a naive distance sensitivity oracle: The first is to do no preprocessing at all and then upon query (u, v, S) compute a shortest u-to-v path in the graph $(V, E \setminus S)$ from scratch. The second extreme is to do all the work in the preprocessing by computing in advance the answer to each one of the $n^{O(|S|)}$ possible queries (u, v, S).

In order to avoid computing the answer to every possible (u, v, S) we would like to generate random subgraphs G_j that have the following property: For every possible (u, v, S) one of the graphs G_j contains *none* of the edges in S but *all* of the edges of a shortest path P between u and v in the graph $(V, E \setminus S)$. This can be achieved with high probability if the path P is short. To handle long paths P, we choose a random subset of vertices B such that with high probability every such long path P decomposes into short subpaths with endpoints in B. These subpaths are captured by the all-pairs distances between vertices of B. To compute these $B \times B$ distances, we use fast matrix multiplication and tweak an algorithm of Yuster and Zwick [Yuster and Zwick 2005] so that it computes all $|B|^2$ distances faster than computing each distance individually. This is all done in the preprocessing.

In query time, in order to find the u-to-v shortest path that avoids S, we construct a new graph G^S with only the vertices $V^S = B \cup \{u, v\}$. The (unbounded) edge-lengths of this graph correspond to the minimal $V^S \times V^S$ distances in all G_js that contain none of the edges in S. The problem then boils down to computing a u-to-v shortest path in this new graph G^S. To do so, we can use a (costly) shortest path algorithm that can handle unbounded (and negative) lengths. We reweigh the *original* graph so that Dijkstra's algorithm can be used instead on any G^S.

Roadmap. We begin in Section 2 with a high level outline of our distance sensitivity oracle, and give a detailed description and analysis in Section 3 leading to the bounds of Theorem 1.2. In Section 4 we show how to use the oracle to solve the replacement paths problem, and in Section 5 we show how to modify the oracle to handle failed vertices rather than edges. We conclude in Section 6.

2. OUTLINE OF THE DISTANCE SENSITIVITY ORACLE

2.1. The Preprocessing

In the preprocessing step, given some $0 < \alpha < 1$ and $f \geq 1$, we first generate the random subgraphs $G_1, \ldots, G_r$ where $r = \tilde{O}(fn^{1-\alpha})$. These graphs are generated independently as follows: Each random subgraph G_j is obtained from G by removing every edge with probability $n^{(\alpha-1)/f}$.

Preparing for short paths. Let F_S denote the set of graphs G_j that *do not* contain any edge from the set S. We would like to have the property that for every possible query (u, v, S) at least one of the graphs $G_j \in F_S$ contains all the edges of a shortest path P between u and v in the graph $(V, E \setminus S)$. This way, upon query (u, v, S), we could report P as the minimal u-to-v shortest path in all $G_j \in F_S$. The query would be fast since with high probability $|F_S|$ is at most $O(f \lg n)$ for any S. It turns out that the property indeed holds for every P that is sufficiently short (shorter than $n^{(1-\alpha)/f}$). The problem

is how to handle the long Ps. For these Ps, the desired property is not guaranteed to hold with high probability. It does however hold for short subpaths of P.

Preparing for long paths. We define an *interval* of a long P as a subpath of P consisting of $n^{(1-\alpha)/f}$ consecutive vertices, so every P induces at most n (overlapping) intervals. We show that with high probability, the edges of any interval induced by P are all present in at least one $G_j \in F_S$. However, we cannot assure that all the intervals of P are in the same G_j. To overcome this, we pick a random subset $B \subseteq V$ of $|B| = \tilde{O}(fn^{1+(\alpha-1)/f})$ vertices. We show that with high probability each of the $O(n^{2f+3})$ possible intervals[2] has at least one vertex in B. This way, every P decomposes into *disjoint* intervals whose endpoints are both in B. Upon query, we will find P using the all-pairs distances between vertices of B in every $G_j \in F_S$. To this end, we precompute the all-pairs $B \times B$ distances for every $G_j \in \{G_1, \ldots, G_r\}$. There are two options (leading to the different construction times in Theorem 1.1): Either we store only the $B \times B$ distances of G_j in a $|B| \times |B|$ matrix B_j, or we store the all-pairs $V \times V$ distances of G_j in an $n \times n$ matrix A_j.

Computing the matrices A_j or B_j and D_j. If we choose to construct the matrix A_j, then we can do so with a classical $O(n^3)$-time all-pairs shortest paths (APSP) algorithm. When the edge-lengths are small integers in $\{-M, \ldots, M\}$ we instead use the APSP $O(M^{0.681}n^{2.575})$-time algorithm of Zwick [Zwick 2002] (with rectangular matrix mult.).

If however we choose to construct the matrix B_j, then we can do so using the algorithm of Yuster and Zwick [Yuster and Zwick 2005]: Given G_j, it constructs in $\tilde{O}(Mn^\omega)$ time, an $n \times n$ matrix D_j which has the property that the distance in G_j between *any* pair of vertices u, v is equal to $\min_{\ell \in V}\{D_j[u, \ell] + D_j[\ell, v]\}$ and can therefore be computed from D_j in $O(n)$ time. The entire $B \times B$ distances can thus be naively computed from D_j in $O(|B|^2 n)$ time. We present an even faster way of doing this. We first extract from D the rows (D_1) and columns (D_2) that correspond to B. We then prove that it is safe to only consider entries in D_1 and D_2 whose absolute value is bounded by $Mn^{(1-\alpha)/f}$. Finally, the distance product $D_1 \star D_2$, that gives us exactly the $B \times B$ distances we need, is computed using the Alon et al. [Alon et al. 1997] algorithm for distance product of matrices with an associated bound on the entries.

This concludes the preprocessing; for every $j = 1, \ldots, r$ our distance sensitivity oracle stores the graphs G_j and either the matrix A_j or the matrices B_j and D_j.

2.2. The Query

Upon query (u, v, S) we seek the shortest u-to-v path P in the graph $(V, E \setminus S)$. We first compute the set F_S of all graphs G_j that do not contain any edge from S. This is done by going over $G_1, \ldots, G_r$ and checking each one in $O(f)$ time. We then compute the u-to-v distance in all $G_j \in F_S$. This finds P in case P is short, and is done in $O(n)$ time if we use the matrix D_j or in $O(1)$ time if we use the matrix A_j. To handle long Ps, we construct a *dense distance graph* G^S: Its set of vertices is $V^S = B \cup \{u, v\}$, and the weight of the edge (x, y) is $\min_{G_j \in F_S}\{\text{distance from } x \text{ to } y \text{ in } G_j\}$. A shortest u-to-v path in G^S will give us the required path P with high probability.

Constructing the graph G^S. To construct G^S, we need the $V^S \times V^S$ distances in every $G_j \in F_S$. If we used A_j then we already have these distances in A_j. However, if we

[2] The number of possible queries (u, v, S) (and thus the number of possible paths P) is $O(n^{2f+2})$ since there are n options for u, n options for v, and n^{2f} options for S (a subset of f edges from a set of $O(n^2)$ edges). Since each of the $O(n^{2f+2})$ paths induces n intervals we have a total of at most $O(n^{2f+3})$ possible intervals.

used B_j and D_j instead, then we have the $B \times B$ distances in B_j and we just need the $\{u\} \times B$ and the $B \times \{v\}$ distances. Each one of these distances is computed in $O(n)$ time using D_j.

To answer the query, all that is left is to compute the u-to-v distance in the graph G^S. Notice that the edge-lengths of G^S may be positive or negative, and they are no longer bounded. Therefore, we can not directly use Dijkstra's single source shortest paths (SSSP) algorithm. To circumvent this, we use the well known method of *feasible price functions* [Johnson 1977] that transforms a shortest-path problem involving positive and negative lengths into one involving only nonnegative lengths, which can then be solved using Dijkstra's SSSP algorithm. We show that the SSSP distances in the original graph G (from any source) serve as a feasible price function for every possible G^S.

We next fill out the missing details and analysis of the above outline. We focus on computing the length of the path P; the actual path can be easily found in the same time bounds.

3. A DISTANCE SENSITIVITY ORACLE

In this section we give a detailed description of our distance sensitivity oracle. Upon query (u, v, S) with $|S| \leq f$, the oracle outputs the length of a shortest u-to-v path that avoids all the edges in the set S. Our goal is to construct an oracle in the bounds of Theorem 1.2.

3.1. The Preprocessing

In preprocessing, we first generate the subgraphs $G_1, \ldots, G_r$ where G_j is obtained from G by removing every edge with probability $n^{(\alpha-1)/f}$. We choose $r = 42fn^{1-\alpha} \lg n$, and since G may have n^2 edges, generating these subgraphs requires $O(rn^2) = \tilde{O}(fn^{3-\alpha})$ time and space. Recall that F_S denotes the set of graphs G_j that *do not* contain any edge in S, and let $f_S = |F_S|$. We start with the following lemma, stating that with high probability f_S is roughly $f \lg n$ for *all* possible S.

LEMMA 3.1. *The probability that* $21f \lg n < f_S < 70f \lg n$ *for all possible* $S \subseteq E$ *(where* $|S| \leq f$*) is at least* $1 - 2/n^f$.

PROOF. We first show that for a specific S, the probability that $f_S > 21f \lg n$ is at least $1 - 1/n^{3f}$. To see this, notice that the expectation of f_S is precisely $E[f_S] = r(n^{(\alpha-1)/f})^f = 42f \lg n$. So by Chernoff's bound (cf. [Alon and Spencer 2000]) we know that

$$Pr[f_S < E[f_S] - 21f \lg n] < e^{\frac{-(21f \lg n)^2}{2E[f_S]}} < \frac{1}{n^{3f}}.$$

We therefore have, by union bound over all possible $O(n^{2f})$ subsets of edges of size at most f, that with high $(1 - 1/n^f)$ probability $f_S > 21f \lg n$. Similarly, by Chernoff's bound (stating that for $a \geq (2/3)E[f_S]$ we have $Pr[f_S < E[f_S] + a] < e^{-2E[f_S]/27}$) we get that

$$Pr[f_S < E[f_S] + 28f \lg n] < e^{-2E[f_S]/27} < \frac{1}{n^{3f}}.$$

So again, by union bound, we have that with high $(1 - 1/n^f)$ probability $f_S < 70f \lg n$ for *all* possible $S \subseteq E$ (where $|S| \leq f$). To conclude, with probability at least $1 - 2/n^f$ we get that $21f \lg n < f_S < 70f \lg n$ for *all* possible $S \subseteq E$ (where $|S| \leq f$). □

Given some specific set of edges S, let P denote a shortest path from u to v in the graph $(V, E \setminus S)$. The path P induces less than n (overlapping) *intervals*. An interval of

P is a subpath consisting of $n^{(1-\alpha)/f}$ consecutive vertices. We say that an interval of P *survives* if all the edges of this interval are present in some $G_j \in F_S$ (recall that F_S is the set of graphs G_j that *do not* contain any edge from the set S). We now show that all the possible intervals survive with high probability as stated by the following lemma. The number of possible intervals is less than n^{2f+3} as each one of the (at most) n^{2f+2} possible queries (u, v, S) corresponds to a path P that induces less than n intervals.

LEMMA 3.2. *The probability that all possible $O(n^{2f+3})$ intervals survive is at least $1 - 3/n^f$.*

PROOF. Consider some specific (u, v, S), and some $G_j \in F_S$. Let P denote a shortest path from u to v in the graph $(V, E \setminus S)$ and let I be some interval of P. The probability that all the $|I| - 1$ edges of I survived in G_j is precisely $(1 - n^{(\alpha-1)/f})^{|I|-1} \geq (1 - n^{(\alpha-1)/f})^{n^{(1-\alpha)/f}-1} > 1/e$. So the probability that I does *not* survive in any $G_j \in F_S$ is less than $(1-1/e)^{f_S}$. If we assume that $f_S > 21f \lg n$ then $(1-1/e)^{f_S} < (1-1/e)^{21f \lg n} < (1/e)^{6f \lg n} < 1/n^{6f}$. In particular, by union bound, we will get that with probability $(1 - 1/n^{4f-3})$ all the possible intervals, whose number is less than n^{2f+3}, survive. By Lemma 3.1, the probability that *all* f_Ss are indeed such that $f_S > 21 \lg n$ is at least $(1-2/n^f)$. We get that the probability that all intervals survive is at least $1-1/n^{4f-3}-2/n^f > 1 - 3/n^f$. □

After establishing that all intervals survive, we need to show that with high probability every interval has at least one vertex in B. We choose B to be a random subset of $6fn^{1+(\alpha-1)/f} \lg n$ vertices of G. Notice that this is where the bound of $f \leq (1 - \alpha) \lg n / \lg \lg n$ is crucial for otherwise we would get $|B| > n$.

LEMMA 3.3. *With probability at least $1 - 1/n^f$ every interval contains some vertex in B.*

PROOF. Since B is chosen randomly, the probability that a specific vertex v does not belong to B is exactly $1 - |B|/n = 1 - 6fn^{(\alpha-1)/f} \lg n$. So an entire interval does not belong to B with probability $(1 - 6fn^{(\alpha-1)/f} \lg n)^{n^{(1-\alpha)/f}} < 1/n^{6f}$. There are at most n^{2f+3} intervals overall, so by union bound we get that with probability at least $1 - 1/n^{4f-3} \geq 1 - 1/n^f$ every interval contains a vertex in B. □

Having generated the graphs $G_1, \ldots, G_r$ and the set B, we now either construct the matrices $A_1, \ldots, A_r$ (where A_j is the $n \times n$ matrix storing the all-pairs distances between all vertices of G_j) or the matrices $B_1, \ldots, B_r$ (where B_j is the $|B| \times |B|$ matrix storing the all-pairs distances between vertices of B in the graph G_j). If we choose to construct the matrix A_j, then we can do so with a classical $O(n^3)$-time APSP algorithm. When the edge-lengths are small integers in $\{-M, \ldots, M\}$ we instead use the APSP $O(M^{0.681}n^{2.575})$-time algorithm of Zwick [Zwick 2002].

COROLLARY 3.4. *The graphs $G_1, \ldots, G_r$ and the matrices $A_1, \ldots, A_r$ can all be constructed in total $\tilde{O}(n^{3-\alpha})$ space, and $O(n^{4-\alpha})$ time for real edge-lengths or $O(M^{0.681}n^{3.575-\alpha})$ time for edge-lengths in $\{-M, \ldots, M\}$.*

If instead we wish to compute $B_1, \ldots, B_r$ and $D_1, \ldots, D_r$, then this can be done faster than computing $A_1, \ldots, A_r$. For two vertices u, v in G_j let $c(u, v)$ denote the least number of edges on a shortest path from u to v in the graph G_j and let $\delta(u, v)$ denote the distance from u to v.

LEMMA 3.5 ([YUSTER AND ZWICK 2005]). *Given an n-vertex graph G_j, the Yuster and Zwick algorithm constructs in $\tilde{O}(Mn^\omega)$ time, an $n \times n$ matrix D_j with the following*

properties: For any pair of vertices u, v there exists a vertex k on a shortest path realizing $c(u,v)$ so that $D_j[u,k] = \delta(u,k)$, $D_j[k,v] = \delta(k,v)$, and $D_j[u,k] + D_j[k,v] = \delta(u,v)$.

In order to compute B_j, we first run the Yuster and Zwick algorithm on G_j constructing the matrix D_j as in Lemma 3.5. Consider the sub-matrix D_j^1 of D_j which consists of taking only the rows that correspond to B. Let D_j^2 be the sub-matrix of D_j that consists only of the columns that correspond to B. It is easy to see that B_j is exactly the distance product $D_j^1 \star D_j^2$ (the distance product $C = A \star B$ of two matrices A and B is defined as $C[i,j] = min_k\{A[i,k] + B[k,j]\}$). To compute the distance product $D_j^1 \star D_j^2$ we use the following result, first stated by Alon et al. [Alon et al. 1997], following a related idea of Yuval [Yuval 1976]. The value $\omega(r,s,t)$ is the matrix multiplication exponent of multiplying an $n^r \times n^s$ matrix with an $n^s \times n^t$ matrix.

LEMMA 3.6 ([ALON ET AL. 1997]). *If A is a $n^r \times n^s$ matrix and B is a $n^s \times n^t$ matrix, both with elements taken from $\{-L, \ldots, L\} \cup \{+\infty\}$, then the distance product $A \star B$ can be computed in $\tilde{O}(Ln^{\omega(r,s,t)})$ time.*

We would like to compute $D_j^1 \star D_j^2$ using the bounds of Lemma 3.6. The problem is that the elements of D_j^1 and D_j^2 are not bounded, in fact L can be as large as Mn. However, we claim that we only need to consider elements of D_j^1 and D_j^1 whose absolute value is less than $L = Mn^{(1-\alpha)/f}$; the rest of the elements can be replaced by $+\infty$. This idea, together with Lemmas 3.5 and 3.6, give the following.

LEMMA 3.7. *Every matrix B_j can be computed in time $\tilde{O}(Mn^\omega + Mn^{(1-\alpha)/f} n^{\omega(1+(\alpha-1)/f, 1, 1+(\alpha-1)/f)})$.*

PROOF. We need to show that any element of D_j^1 and D_j^2 whose absolute value is greater than $Mn^{(1-\alpha)/f}$ can be thought of as $+\infty$. To see this, consider a shortest path between two vertices $x, y \in B$. Recall that we assume every *interval* of $n^{(1-\alpha)/f}$ vertices has at least one vertex in B. Therefore, we are only interested in the x-to-y shortest path if the number of its edges $c(x,y) \leq n^{(1-\alpha)/f}$. In this case, by Lemma 3.5, there must be some $k \leq n$ such that $c(x,k) \leq n^{(1-\alpha)/f}$, and $c(k,y) \leq n^{(1-\alpha)/f}$, and $D_j[x,k] + D_j[y,k]$ is the length of the x-to-y shortest path. Since the absolute value of every edge-length is bounded by M, we get that $D_j[x,k] \leq Mn^{(1-\alpha)/f}$ and $D_j[y,k] \leq Mn^{(1-\alpha)/f}$. So the corresponding entries in D_j^1 and D_j^2 are bounded by $Mn^{(1-\alpha)/f}$. □

By Lemma 3.7, we can compute all the B_j-matrices in total time

$$\tilde{O}(rMn^\omega + rMn^{(1-\alpha)/f} n^{\omega(1+(\alpha-1)/f, 1, 1+(\alpha-1)/f)}).$$

Recalling that $r = \tilde{O}(fn^{1-\alpha})$, that $f = \tilde{O}(1)$, that $\alpha \leq 1$, that $\omega \geq 2$, and that $\omega(x,1,x) \leq \omega x + 1 - x$ (See, e.g., Huang and Pan [Huang and Pan 1998]), we have that $\tilde{O}(rMn^\omega + rMn^{(1-\alpha)/f} n^{\omega(1+(\alpha-1)/f, 1, 1+(\alpha-1)/f)}) = \tilde{O}(Mn^{1-\alpha+\omega})$. The size of each B_j is $|B|^2$ so the total space required for all B_js is $O(r \cdot |B|^2) = \tilde{O}(n^{3-\alpha+(2\alpha-2)/f})$. In the process, we also generate all the matrices D_j. The size of each D_j is n^2 so the total space required for all D_js is $O(r \cdot n^2) = \tilde{O}(n^{3-\alpha})$. We thus conclude the description of the preprocessing with the following corollary.

COROLLARY 3.8. *The graphs $G_1, \ldots, G_r$, the matrices $B_1, \ldots, B_r$, and the matrices $D_1, \ldots, D_r$ can all be constructed in total time $\tilde{O}(Mn^{1-\alpha+\omega})$ and space $\tilde{O}(n^{3-\alpha})$.*

3.2. The Query

Upon query (u, v, S) we seek the shortest u-to-v path P in the graph $(V, E \setminus S)$. We first identify the set F_S of all graphs G_j that do not contain any edge from S. This is done by going over $G_1, \ldots, G_r$ and checking each one in $O(f)$ time for a total of $O(fr) = \tilde{O}(n^{1-\alpha})$ time. By Lemma 3.1, we get that $f_S = |F_S| = \Theta(f \lg n)$. We then compute the u-to-v distance in every $G_j \in F_S$. This finds P in case P is short and can be done in a total of $O(f_S)$ time using the A_j matrices or in $O(nf_S) = \tilde{O}(n)$ time using the D_j matrices (by Lemma 3.5).

To handle long Ps, we construct a *dense distance graph* G^S: Its set of vertices is $V^S = B \cup \{u, v\}$, and the length of the edge (x, y) is $\min_{G_j \in F_S}\{\text{distance from } x \text{ to } y \text{ in } G_j\}$. A shortest u-to-v path in G^S will give us the required path P with high probability.

Constructing the graph G^S. To construct G^S, we need the distances in every $G_j \in F_S$ between any two vertices in V^S. Once we have these $V^S \times V^S$ distances, each edge-length in G^S requires examining f_S values for a total of $O(f_S|B|^2) = \tilde{O}(n^{2-2(1-\alpha)/f})$ time.

If we used the A_j matrices then we already have the $V^S \times V^S$ distances stored and we do not need to spend any additional time. If however we used the B_j matrices then we already have the distances between any two vertices in B and we have also just computed the u-to-v distance in each $G_j \in F_S$. What is left is to compute the distance from u to all vertices of B and from all vertices of B to v. Again, using Lemma 3.5, each one of these $O(|B|)$ distances is computed in $O(n)$ time from D_j. The overhead of using the B_j (rather than A_j) matrices over all $G_j \in F_S$ is thus $O(n|B|f_S) = \tilde{O}(n^{2-(1-\alpha)/f})$ time.

Having constructed the graph G^S, all that is left is to compute the u-to-v distance in G^S. Notice that the edge-lengths of G^S are no longer bounded and may be positive or negative. Therefore, we can not directly use Dijkstra's single source shortest paths (SSSP) algorithm (that can handle unbounded lengths but can not handle negative lengths). We may run Goldberg's SSSP algorithm (which can handle large integral negative lengths) but this would be too costly. Instead, we use the well known method of *feasible price functions* in order to transform the edge-lengths of G^S to be nonnegative and then use Dijkstra's SSSP algorithm.

Computing the u-to-v distance in G^S. For a directed graph $G = (V, E)$ with (possibly negative) edge-lengths $w(\cdot)$, a *price function* is a function ϕ from the vertices of G to the reals. For an edge (u, v), its *reduced length with respect to* ϕ is $w_\phi(u, v) = w(u, v) + \phi(u) - \phi(v)$. A price function ϕ is *feasible* if $w_\phi(u, v) \geq 0$ for *all* edges $(u, v) \in E$. The idea behind feasible price functions is that for any two vertices $s, t \in V$, for any s-to-t path P, $w_\phi(P) = w(P) + \phi(s) - \phi(t)$. This shows that an s-to-t path is shortest with respect to $w_\phi(\cdot)$ iff it is shortest with respect to $w(\cdot)$. Moreover, the s-to-t distance with respect to the original lengths $w(\cdot)$ can be recovered by adding $\phi(t) - \phi(s)$ to the s-to-t distance with respect to $w_\phi(\cdot)$.

The most natural feasible price function comes from single-source distances. Let s be a new vertex added to G with an edge from s to every other vertex of G having weight 0. Let $d(v)$ denote the distance from s to vertex $v \in G$. Then for every edge $(u, v) \in E$, we have that $d(v) \leq d(u) + w(u, v)$, so $w_d(u, v) \geq 0$ and thus $d(\cdot)$ is feasible. This means that knowing $d(\cdot)$, we can now use Dijkstra's SSSP algorithm on G (with reduced lengths) from any source we choose and obtain the SSSP with respect to the original G. However, we would like to run Dijkstra not on G but on G^S. The following lemma states that we can use the same price function $d(v)$ on all possible G^Ss.

LEMMA 3.9. *The function $d(\cdot)$ is a feasible price function for every possible G^S.*

PROOF. Consider an edge (x,y) in G^S. We need to prove that $w_d(x,y) \geq 0$. We know that $w(x,y)$ is the length of a shortest x-to-y path in all $G_j \in F_S$. In particular, $w(x,y)$ is the length of some (not necessarily shortest) x-to-y path P in G. Consider the s-to-y path in G that is composed of a shortest s-to-x path in G and the path P. The length of this s-to-y path is $d(x) + w(x,y)$ and therefore $d(y) \leq d(x) + w(x,y)$ so $w_d(x,y) = w(x,y) + d(x) - d(y) \geq 0$. □

Lemma 3.9 suggests that during preprocessing-time we also compute the distances $d(v)$ in G using SSSP from s, and then during query-time use $d(\cdot)$ as a price function. This single SSSP computation only adds $O(n^3)$ time to the preprocessing (using the Bellman-Ford SSSP) or $\tilde{O}(Mn^\omega)$ time if the edge-lengths are in $\{-M,\ldots,M\}$ (using the Yuster and Zwick SSSP). Observe that these running times are negligible with respect to the preprocessing time that we have already spent. Then, in query-time, we run Dijkstra's SSSP on G^S from u using the price function $d(\cdot)$ in $O(|B|^2) = \tilde{O}(n^{2-2(1-\alpha)/f})$ time.

To conclude, the dominating term in the query time is $\tilde{O}(n^{2-2(1-\alpha)/f})$ if we use the A_j matrices and $\tilde{O}(n^{2-(1-\alpha)/f})$ if we use the B_j matrices. Together with Corollaris 3.4 and 3.8 this gives the bounds of Theorem 1.1.

4. THE REPLACEMENT PATHS ALGORITHM

In this section, we show how our distance sensitivity oracle for avoiding failed edges can be made to solve the replacement paths problem in the bounds of Corollary 1.2.

Recall that in the *replacement paths problem* we are given a directed graph $G = (V,E)$ with positive and negative edge-lengths and two distinct vertices u and v. Let $P = (u = v_0, v_1, \ldots, v_k = v)$ be a shortest path from u to v where $1 \leq k < n$ (we assume no negative cycles; such cycles will be detected if they exist). We wish to compute paths $P_1,\ldots,P_k$ such that P_i is a shortest path from u to v in the graph $G \setminus e_i$ where e_i is the edge (v_{i-1}, v_i).

As a first step, we compute some shortest u-to-v path in G in order to identify $e_1,\ldots,e_k$. This can be done in $\tilde{O}(Mn^\omega)$ time using the SSSP algorithm of Yuster and Zwick. We then build a distance sensitivity oracle for G as in the previous section. We use the construction with the B_j matrices, but we make one small change in the preprocessing stage: After we randomly choose the set of vertices B we explicitly add u and v to B. This does not change the construction time which by Theorem 1.1 (with $f = 1$) requires $\tilde{O}(Mn^{1-\alpha+\omega})$ time.

Finally, we ask the oracle queries $(u, v, \{e_i\})$ for $i = 1,\ldots,k$. Naively, using Theorem 1.1, each query requires $\tilde{O}(n^{2-(1-\alpha)/f}) = \tilde{O}(n^{1+\alpha})$ time so all queries take $\tilde{O}(n^{2+\alpha})$ time. However, since now u and v are in B, we don't need to compute the $\{u\} \times B$ and $B \times \{v\}$ distances in query-time. This was the $\tilde{O}(n^{2-(1-\alpha)/f})$ bottleneck and without it the query takes $\tilde{O}(n^{2-2(1-\alpha)/f}) = \tilde{O}(n^{2\alpha})$ time.

Total time complexity. From the above description, the total time complexity of our replacement paths algorithm is

$$\tilde{O}(Mn^\omega + Mn^{1-\alpha+\omega} + n^{1+2\alpha}).$$

To get the desired bounds we will choose the optimal value for α. First notice that we must insist on $M \leq n^{3-\omega} = n^{0.624}$ for otherwise the time complexity is worse than the trivial $O(n^3)$ solution. Setting the second and third terms to be equal determines the best choice of $\alpha = \frac{\omega}{3} + \frac{\lg M}{3 \lg n}$ implying that $Mn^{1-\alpha+\omega} = n^{1+2\alpha} = M^{\frac{2}{3}} n^{1+\frac{2}{3}\omega}$. Notice that indeed $\alpha \leq 1$ since $M \leq n^{3-\omega}$. The total time complexity is therefore $\tilde{O}(Mn^\omega + M^{\frac{2}{3}} n^{1+\frac{2}{3}\omega})$ thus proving Corollary 1.2.

5. AVOIDING VERTICES RATHER THAN EDGES

In this section, we explain how our distance sensitivity oracle can be made to work in the same bounds of Theorem 1.1 for the case of avoiding vertices. That is, when in the query (u, v, S) the set $S \subseteq V \setminus \{u, v\}$ is a set of *vertices* and we seek a shortest u-to-v path in the graph obtained from G by removing all vertices in S and their adjacent edges. We follow the exact same lines as Section 3 and only highlight the differences for the case of avoiding vertices.

5.1. The Preprocessing

In preprocessing, we generate the subgraphs $G_1, \ldots, G_r$ with $r = 42fn^{1-\alpha} \lg n$. This time, G_j is obtained from G by removing every *vertex* (and its adjacent edges) with probability $n^{(\alpha-1)/f}$. Generating these subgraphs requires $O(rn) = \tilde{O}(fn^{2-\alpha})$ time and space.

The set F_S now denotes the set of graphs G_j that do not contain any *vertex* in S. The statement of Lemma 3.1 is still correct (after replacing $S \subseteq E$ with $S \subseteq V$). In particular, the expectation of f_S is still $E[f_S] = r(n^{(\alpha-1)/f})^f = 42f \lg n$. The only minor change in the proof is where we use union bound over all possible $O(n^{2f})$ subsets of edges of size at most f. We now only have to union bound over all possible $O(n^f)$ subsets of vertices of size at most f. This only helps in the analysis.

Given some specific set of vertices S and a pair of vertices $u, v \notin S$, let P denote a shortest path from u to v in the graph $(V \setminus S, E)$. The path P induces at most $O(n)$ (overlapping) *intervals*. An interval of P, just as in Section 3, is a subpath consisting of $n^{(1-\alpha)/f}$ consecutive vertices, and is said to *survive* if all the vertices of this interval are present in some $G_j \in F_S$.

The total number of possible intervals is now only $O(n^{f+3})$ as each one of the $O(n^{f+2})$ possible queries (u, v, S) corresponds to a path P that induces $O(n)$ intervals. The statement of Lemma 3.2 is still true, we just need to change $O(n^{2f+3})$ to $O(n^{f+3})$ and $(V, E \setminus S)$ to $(V \setminus S, E)$. In the proof of this lemma, we make the minor change of claiming that "The probability that all the $|I|$ vertices of I survived in G_j is precisely $(1 - n^{(\alpha-1)/f})^{|I|} \geq (1 - n^{(\alpha-1)/f})^{n^{(1-\alpha)/f}} = 1/e$". The rest of the proof is identical except for the union bound which is now over $O(n^{f+3})$ possible intervals (and not $O(n^{2f+3})$). This only helps in the analysis.

Lemma 3.3 and the rest of the preprocessing is exactly like in Section 3.

5.2. The Query

Upon query (u, v, S) we seek the shortest u-to-v path P in the graph $(V \setminus S, E)$. As before, we first identify the set F_S of all graphs G_j that do not contain any vertex from S. This is done by going over $G_1, \ldots, G_r$ and checking each one in $O(f)$ time for a total of $O(fr) = \tilde{O}(n^{1-\alpha})$ time.

We then wish to compute the u-to-v distance in every $G_j \in F_S$. Notice that u and v are not necessarily present in all $G_j \in F_S$. However, by Lemma 3.2, all the vertices of an interval survive in some G_j. In particular, the vertex u (resp. v) survives along with the entire interval composed of a prefix (resp. suffix) of P starting (resp. ending) with u (resp. v). This finds P in case P is short (i.e., P is composed of a single interval containing both u and v) and as before it can be done in total $O(f_S)$ time using the A_j matrices or in $\tilde{O}(n)$ time using the D_j matrices.

To handle long Ps, we construct a *dense distance graph* G^S: Its set of vertices is now $V^S = B \cup \{u, v\} \setminus S$ as opposed to $B \cup \{u, v\}$. Notice that some of the vertices of B may now be removed by S. However, by Lemmas 3.2 and 3.3 we know that every interval survives entirely and so the vertex of B that hits this interval also survives.

The rest of the query stage is handled exactly as in Section 3.

6. CONCLUDING REMARKS

We have presented a distance sensitivity oracle that is constructed in sub-cubic time and upon query (u, v, S) with $|S| = O(\lg n / \lg \lg n)$ returns in sub-quadratic time the length of a shortest u-to-v path that avoids all the edges or vertices in the set S. We have also presented a sub-cubic time algorithm for the replacement paths problem in directed graphs with small integral edge-lengths.

Even though our oracle's query time is rather slow, it is still the fastest known for $f > 2$ faults. In fact, for vertex faults, even if we are willing to settle for approximate distances, the only known oracle of Chechik et al. [Chechik et al. 2010] that works for avoiding f edges does not work for avoiding f vertices. Improving our query time, even at the cost of allowing approximation, is an important open problem.

As for the replacement paths problem, the main open question is whether we can solve it in sub-cubic time on directed graphs with unbounded (i.e, independent of M) weights. Even an $O(n^{3-\delta} poly(\lg M))$ time solution for any $\delta > 0$ would be very interesting since by a recent result of Vassilevska-Williams and Williams [Vassilevska-Williams and Williams 2010] it will imply that many other important problems have an $O(n^{3-\delta} poly(\lg M))$ solution. These problems include the all-pairs shortest paths problem on weighted digraphs, detecting if a weighted graph has a triangle of negative total edge weight, finding a minimum weight cycle in a graph of non-negative edge weights, and more.

REFERENCES

N. Alon, Z. Galil, and O. Margalit. 1997. On the exponent of the all pairs shortest path problem. *J. Comput. System Sci.* 54 (1997), 255–262.

N. Alon and J.H. Spencer. 2000. *The probabilistic method* (2nd ed.). Wiley-Interscience.

A. Bernstein. 2010. A Nearly Optimal Algorithm for Approximating Replacement Paths and k Shortest Simple Paths in General Graphs. In *Proc. of the 21st ACM-SIAM Symposium On Discrete Algorithms (SODA)*. 742–755.

A. Bernstein and D. Karger. 2008. Improved distance sensitivity oracles via random sampling. In *Proc. of the 19th ACM-SIAM Symposium On Discrete Algorithms (SODA)*. 34–43.

A. Bernstein and D. Karger. 2009. A nearly optimal oracle for avoiding failed vertices and edges. In *Proc. of the 41st ACM Symposium on Theory of Computing (STOC)*. 101–110.

S. Chechik, M. Langberg, D. Peleg, and L. Roditty. 2010. f -Sensitivity Distance Oracles and Routing Schemes. In *Proceedings of the 18th annual European Symposium on Algorithms (ESA)*. 84–96.

C. Demetrescu and G.F. Italiano. 2004. Experimental analysis of dynamic all pairs shortest path algorithms. In *Proc. of the 15th ACM-SIAM Symposium On Discrete Algorithms (SODA)*. 362–371.

C. Demetrescu, M. Thorup, R. Chowdhury, and V. Ramachandran. 2008. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.* 37, 5 (2008), 1299–1318.

R. Duan and S. Pettie. 2009. Dual-failure distance and connectivity oracles. In *Proc. of the 20th ACM-SIAM Symposium On Discrete Algorithms (SODA)*. 506–515.

Y. Emek, D. Peleg, and L. Roditty. 2008. A near-linear time algorithm for computing replacement paths in planar directed graphs. In *Proc. of the 19th ACM-SIAM Symposium On Discrete Algorithms (SODA)*. 428–435.

D. Eppstein. 1999. Finding the k shortest paths. *SIAM J. Comput.* 28 (1999), 652–673.

Z. Gotthilf and M. Lewenstein. 2009. Improved algorithms for the k simple shortest paths and the replacement paths problems. *Inform. Process. Lett.* 109, 7 (2009), 352–355.

J. Hershberger and S. Suri. 2001. Vickrey prices and shortest paths: what is an edge worth?. In *Proc. of the 42nd annual symposium on Foundations Of Computer Science (FOCS)*. 252–259.

J. Hershberger, S. Suri, and A. Bhosle. 2003. On the difficulty of some shortest path problems. In *Proc. of the 20th Symposium on Theoretical Aspects of Computer Science (STACS)*. 343–354.

X. Huang and V. Pan. 1998. Fast rectangular matrix multiplications and applications. *Journal of Complexity* 14 (1998), 257–299.

D.B. Johnson. 1977. Efficient algorithms for shortest paths in sparse networks. *J. ACM* 1 (1977), 1–14.

D. Karger, D. Koller, and S. J. Phillips. 1993. Finding the hidden path: Time bounds for all-pairs shortest paths. *SIAM J. Comput.* 22, 6 (1993), 1199–1217.

P. Klein, S. Mozes, and O. Weimann. 2009. Shortest Paths in Directed Planar Graphs with Negative Lengths: a Linear-Space $O(nlog^2n)$-Time Algorithm. In *Proc. of the 20th ACM-SIAM Symposium On Discrete Algorithms (SODA)*. 236–245.

E.L. Lawler. 1972. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science* 18 (1972), 401–405.

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

E. Nardelli, G. Proietti, and P. Widmayer. 2001. A faster computation of the most vital edge of a shortest path. *Inform. Process. Lett.* 79, 2 (2001), 81–85.

E. Nardelli, G. Proietti, and P. Widmayer. 2003. Finding the most vital node of a shortest path. *Theoretical Computer Science* 296, 1 (2003), 167–177.

N. Nisan and A. Ronen. 2001. Algorithmic mechanism design. *Games and Economic Behavior* 35 (2001), 166–196.

L. Roditty. 2007. On the k-simple shortest paths problem in weighted directed graphs. In *Proc. of the 18th ACM-SIAM Symposium On Discrete Algorithms (SODA)*. 920–4928.

L. Roditty and U. Zwick. 2004. A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In *Proc. of the 36th ACM Symposium on Theory of Computing (STOC)*. 184–191.

L. Roditty and U. Zwick. 2005. Replacement Paths and k Simple Shortest Paths in Unweighted Directed Graphs. In *Proc. of the 32nd annual International Colloquium on Automata, Languages and Programming (ICALP)*. 249–260.

M. Thorup. 1999. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM* 46 (1999), 362–394.

M. Thorup. 2004. Fully-Dynamic All-Pairs Shortest Paths: Faster and Allowing Negative Cycles. In *Proc. of the 9th Scandinavian Workshop on Algorithm Theory (SWAT)*. 384–396.

M. Thorup. 2005. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proc. of the 37th ACM Symposium on Theory of Computing (STOC)*. 112–119.

V. Vassilevska-Williams. 2011. Faster Replacement Paths. In *Proc. of the 22nd ACM-SIAM Symposium On Discrete Algorithms (SODA)*. 1337–1346.

V. Vassilevska-Williams and R. Williams. 2010. Subcubic Equivalences Between Path, Matrix, and Triangle Problems. In *Proceedings of the 51st annual symposium on Foundations Of Computer Science (FOCS)*. 645–654.

O. Weimann and R. Yuster. 2010. Replacement Paths via Fast Matrix Multiplication. In *Proceedings of the 51st annual symposium on Foundations Of Computer Science (FOCS)*. 655–662.

C. Wulff-Nilsen. 2010. Solving the Replacement Paths Problem for Planar Directed Graphs in $O(n \log n)$ Time. In *Proc. of the 21st ACM-SIAM Symposium On Discrete Algorithms (SODA)*. 756–765.

J.Y. Yen. 1971. Finding the K shortest loopless paths in a network. *Management Science* 17 (1971), 712–716.

R. Yuster and U. Zwick. 2005. Answering distance queries in directed graphs using fast matrix multiplication. In *Proc. of the 46th annual symposium on Foundations Of Computer Science (FOCS)*. 389–396.

G. Yuval. 1976. An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications. *Inform. Process. Lett.* 4 (1976), 155–156.

U. Zwick. 2002. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM* 49 (2002), 289–317.