Replacement Paths via Fast Matrix Multiplication

Oren Weimann
Department of Computer Science and Applied Mathematics
The Weizmann Institute of Science
Rehovot, Israel
oren.weimann@weizmann.ac.il

Raphael Yuster
Department of Mathematics
University of Haifa
Haifa, Israel
raphy@math.haifa.ac.il

***Abstract*—Let G be a directed edge-weighted graph and let P be a shortest path from s to t in G. The replacement paths problem asks to compute, for every edge e on P, the shortest s-to-t path that avoids e.**

Apart from approximation algorithms and algorithms for special graph classes, the naive solution to this problem – removing each edge e on P one at a time and computing the shortest s-to-t path each time – is surprisingly the only known solution for directed weighted graphs, even when the weights are integrals. In particular, although the related shortest paths problem has benefited from fast matrix multiplication, the replacement paths problem has not, and still required cubic time.

For an n-vertex graph with integral edge-lengths between -M and M, we give a randomized algorithm that uses fast matrix multiplication and is sub-cubic for appropriate values of M. We also show how to construct a distance sensitivity oracle in the same time bounds. A query (u,v,e) to this oracle requires sub-quadratic time and returns the length of the shortest u-to-v path that avoids the edge e. In fact, for any constant number of edge failures, we construct a data structure in sub-cubic time, that answer queries in sub-quadratic time. Our results also apply for avoiding vertices rather than edges.

I. Introduction

In a network G where edges occasionally fail, a shortest path P between two vertices s and t may have to change accordingly. The *replacement paths* problem asks to compute, for every edge e in P, the shortest s-to-t 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 [1], [2]. 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 [3]. The *all pairs* generalization of replacement paths (also called *distance sensitivity oracles* in the literature) asks to construct a data structure that upon query (u, v, e) outputs the length of a shortest u-to-v path that avoids the edge e. Other generalizations include avoiding more than one edge, or avoiding vertices rather than edges.

Replacement Paths: The naive solution to the replacement paths problem is to remove each edge e on P, one at a time, and compute the shortest s-to-t path each time. This can be done in $O(mn + n^2 \log n)$ time. Except for the slight improvement of Gotthilf and Lewenstein [4] to $O(mn + n^2 \log\log n)$, assuming non-negative weights, no faster algorithms are known for weighted, directed graphs. Similarly, the fastest algorithm for finding k shortest simple paths, given by Yen [5] and Lawler [6], uses k replacement path computations and therefore runs in $O(k(mn + n^2 \log\log n))$ time. Hershberger et al. [7] gave an $\Omega(m\sqrt{n})$ lower bound for both these problems in the path-comparison model of Karger et al. [8].

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. [9] presented an $\tilde{O}(m)$ time algorithm for the replacement paths problem, that was later extended to avoid vertices rather than edges [10]. Nardelli et al. [11] 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 [12]. For directed *unweighted* graphs, Roditty and Zwick [13] gave an $\widetilde{O}(m\sqrt{n})$ time randomized algorithm. Finally, for the class of directed *planar* graphs, Emek et al. [14] gave an $O(n \log^3 n)$ time algorithm, improved in [15] and later in [16] to $O(n \log n)$.

The second way of overcoming the $o(mn)$ bound was to settle for *approximate* distances. Roditty [17] 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 [18] 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 path algorithm is a $(1 + \varepsilon)$-approximation running in $\tilde{O}(m \log(nC/c)/\varepsilon)$ time, where C is the largest edge length in the graph and c is the smallest.

All Pairs Replacement Paths: The more general *all pairs* replacement paths problem asks to construct a *distance*

sensitivity oracle for answering queries (u, v, e) seeking the length of the shortest u-to-v path that avoids the edge e. For directed weighted graphs, Demetrescu et al. [19] presented a distance sensitivity oracle that can be constructed in $\tilde{O}(mn^2)$ time and $O(n^2 \log n)$ space, has constant query time, and can also handle queries for avoiding vertices rather than edges. Bernstein and Karger improved the preprocessing to $O(n^2\sqrt{m})$ [20] and then further to $\tilde{O}(mn)$ [21], with unchanged constant query time. For the case of two failures (vertices or edges) Duan and Pettie [22] construct an oracle in polynomial time and $\tilde{O}(n^2)$ space, whose query time is $O(\log n)$. For more than two failures, Chechick et al. [23] showed that if we are willing to settle for approximate distances then an oracle constructed in polynomial time can handle f failures in $\tilde{O}(f)$ query time.

Finally, we mention the (harder) related problem of maintaining all pairs shortest path queries while updating the graph (deleting and inserting edges and vertex). Demetrescu and Italiano [24] presented a data structure to maintain all pairs shortest path queries in $O(1)$ time with $\tilde{O}(n^2)$ amortized update. Their algorithm was slightly improved by Thorup [25]. For *unweighted undirected* graphs, Roditty and Zwick [26] 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 [27] obtained $O(n^{2.75})$ *worst case* updates.

Our Results: Although several algorithms exploiting fast matrix multiplication are known for *shortest path* problems it was unknown how to use fast matrix multiplication for the tightly-connected *replacement paths* problem. We show how to exploit fast matrix multiplication for the replacement paths problem in directed graphs with integral (positive and negative) edge lengths. Our first main result is a randomized algorithm whose bounds are given by the following theorem.

Theorem 1 *The replacement paths problem can be solved in $\tilde{O}(Mn^{1+\frac{2}{3}\omega}) = O(Mn^{2.584})$ time with very high probability on an n-vertex directed graph with integral edge-lengths in $\{-M, \ldots, M\}$.*

There are a few things to notice about the bounds of Theorem 1. Using the current matrix multiplication exponent $\omega < 2.376$ [28] we get a bound of $O(n^{2.584})$ for fixed M, and sub-cubic time for any $M < n^{0.416}$. A surprising outcome is that if $\omega < 2.25$, as may turn out to be the case, we get, for general graphs, better bounds than the Roditty-Zwick $O(n^{2.5})$ algorithm that can only handle *unweighted* graphs, or graphs with small positive weights. In particular, if $\omega = 2$ we get $O(Mn^{2.333})$.

We further show how to extend our method to the all pairs replacement paths problem, by constructing a distance sensitivity oracle[1] with sub-cubic construction and sub-quadratic (notably, sub-Dijkstra) query time as stated by the following theorem.

Theorem 2 *For any $0 < \alpha < 1$, a distance sensitivity oracle can be constructed in $\tilde{O}(Mn^{1-\alpha+\omega})$ time on an n-vertex directed graph with integral edge-lengths in $\{-M, \ldots, M\}$. A query (u, v, e) to this oracle requires $\tilde{O}(n^{1+\alpha})$ time and returns the length of the shortest u-to-v path that avoids the edge e.*

Using the same ideas of Theorem 2 we can get a more general distance sensitivity oracle that can handle multiple failures with the following bounds.

Corollary 1 *For any $0 < \alpha < 1$, a distance sensitivity oracle for avoiding f edges, can be constructed in $\tilde{O}(Mn^{1-\alpha+\omega})$ time on an n-vertex directed graph with integral edge-lengths in $\{-M, \ldots, M\}$. A query (u, v, S) to this oracle requires $\tilde{O}(n^{2-(1-\alpha)/f})$ time and returns the length of the shortest u-to-v path that avoids all the edges in S.*

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.

Technique: In contrast to the naive replacement paths algorithm that performs $O(n)$ shortest path computations, we compute shortest paths on $o(n)$ random subgraphs. We start with random subgraphs G_j that with very high probability capture all *short* replacement paths. We then choose a random subset of vertices B such that with very high probability any long replacement path in G_j 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 [29] so that it computes all $|B|^2$ distances faster than computing each distance individually. Solving the replacement paths problem then boils down to computing an s-to-t shortest path in a new graph with B vertices and (unbounded) edge-lengths corresponding to the minimal $B \times B$ distances between a few G_js. To compute the s-to-t shortest path in this new graph, we can use a (costly) shortest path algorithm that can handle unbounded (and negative) lengths. We show how to reweigh the graph so that Dijkstra's algorithm can be used instead.

II. The Replacement Paths Algorithm

In the *replacement paths problem* we are given a directed graph $G = (V, E)$ with integral (and possibly negative)

[1] We slightly abuse the term *oracle* here as oracles are usually thought of as having constant or logarithmic query time.

edge-lengths in $\{-M, \ldots, M\}$ and two distinct vertices s and t. Let $P = (s = v_0, v_1, \ldots, v_k = t)$ be a shortest path from s to t 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 s to t in the graph $G \backslash e_i$ where e_i is the edge (v_{i-1}, v_i). In this section, we show how to solve the replacement paths problem in the bounds of Theorem 1. We begin with a high level outline of our replacement paths algorithm. We then give a detailed description and analysis.

A. Outline of the Algorithm.

The naive solution to the replacement paths problem would be to consider the subgraphs $G \setminus e_i$ for $i = 1, \ldots, k$, and compute an s-to-t shortest path in each of them. As the edge lengths are bounded by M, every s-to-t path can be found in $\tilde{O}(Mn^\omega)$ time using the single-source shortest-paths (SSSP) algorithm of Yuster-Zwick [29], or in $\tilde{O}(\sqrt{n}m) = \tilde{O}(n^{2.5})$ time with the SSSP algorithm of Goldberg [30]. However, k executions of an SSSP algorithm are too costly as k can be $O(n)$.

The Random subgraphs G_j: Instead of $k = O(n)$ graphs, we wish to construct $r = \tilde{O}(n^{1-\alpha})$ graphs for some $0 < \alpha < 1$ to be chosen later. These graphs, $G_1, \ldots, G_r$, are random subgraphs of G that are generated independently as follows: Each random subgraph G_j is obtained from G by removing every edge with probability $n^{\alpha-1}$.

Let F_e denote the set of graphs G_j that *do not* contain the edge e. We would like to have the property that with very high probability, for every i, the edges of P_i are all present in at least one $G_j \in F_{e_i}$. This way, we could run an SSSP algorithm on every G_j and report P_i as the minimal s-to-t shortest path in all $G_j \in F_{e_i}$. It turns out, that by choosing the appropriate r, the property holds for every P_i that is sufficiently short (shorter than $n^{1-\alpha}$). The problem is how to handle the long P_is. For these P_is, the desired property is not guaranteed to hold with high probability. It does however hold for short subpaths of P_i.

We define an *interval* as a subpath of P_i consisting of $n^{1-\alpha}$ consecutive vertices, so every P_i induces at most n (overlapping) intervals for a total of n^2 intervals. We show that with very high probability, the edges of any interval induced by P_i are all present in at least one $G_j \in F_{e_i}$. However, we cannot assure that all the intervals of P_i are in the same G_j. To overcome this, we pick a random subset $B \subseteq V$ of $\tilde{O}(n^\alpha)$ vertices and make sure that $s, t \in B$. We show that with very high probability each of the n^2 possible intervals has at least one vertex in B. This way, every P_i decomposes into *disjoint* intervals whose endpoints are both in B.

For every $i = 1, \ldots, k$ we construct the *dense distance graph* G_B^i as follows: Its set of vertices is B, and the weight of the edge (u, v) is the length of the shortest u-to-v path in all $G_j \in F_{e_i}$. The shortest s-to-t path in G_B^i will give us the replacement path P_i. We are thus left with two challenges: How to construct the G_B^is, and how to compute the shortest s-to-t path in every G_B^i.

The Dense distance graphs G_B^i: Constructing the G_B^is boils down to computing, for every G_j the distance between any two vertices in B. Naturally, this can be achieved by computing all-pairs shortest-paths on G_j, but notice that we are only interested in the $B \times B$ subset shortest paths. The algorithm of Yuster-Zwick [29] performs this task faster. It constructs, in $\tilde{O}(Mn^\omega)$ time, an $n \times n$ matrix D which has the property that the distance between *any* pair of vertices u, v is equal to $\min_{\ell \in V}\{D[u, \ell] + D[\ell, v]\}$ and can therefore be computed from D in $O(n)$ time. The entire $B \times B$ distances can thus be naively computed from D 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 $Mn^{1-\alpha}$. 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. [31] algorithm for distance product of matrices with an associated bound on the entries, which uses a related idea of Yuval [32].

s-to-t shortest paths in G_B^i: We are left with the final challenge of computing the s-to-t shortest paths in every G_B^i. Notice that the edge-lengths of G_B^i may be positive or negative, but they are no longer bounded. Therefore, to compute the shortest s-to-t paths, we may run Goldberg's SSSP algorithm multiple times, once for each G_B^i. This algorithm can handle large negative lengths. However, the similarity between the various G_B^is suggests a more efficient computation using the well known method of *reduced lengths*.

This method is useful in transforming 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. Traditionally, the edges of a given graph G are first reweighed so they become nonnegative. Then, multiple runs of Dijkstra are performed on the *same* graph G (now with nonnegative edge-lengths) but with different sources s. We introduce a variant of this method, where we reduce the lengths once and then Dijkstra is performed multiple times but on *different* graphs. Namely, once on each G_B^i.

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

As a first step, we compute some shortest s-to-t 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-Zwick.

B. The Random Subgraphs G_j.

We generate the subgraphs $G_1, \ldots, G_r$ where G_j is obtained from G by removing every edge with probability $n^{\alpha-1}$. We choose $r = 42n^{1-\alpha}\log n$, and since G may have n^2 edges, generating these subgraphs requires $O(rn^2) = \tilde{O}(n^{3-\alpha})$ time. Recall that F_e denotes the set of graphs G_j that *do not* contain the edge e, and let $f_e = |F_e|$. We start with the following lemma, stating that with very high probability f_e is roughly equal to $\log n$ for *all* $e \in E$.

Lemma 1 *The probability that* $21\log n < f_e < 70\log n$ *for all* $e \in E$ *is at least* $1 - 2/n$.

Proof: We first show that for a single edge e, the probability that $f_e > 21\log n$ is at least $1 - 1/n^3$. To see this, notice that the expectation of f_e is precisely $E[f_e] = rn^{\alpha-1} = 42\log n$. So by Chernoff's bound (cf. [33]) we know that

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

We therefore have, by union bound, that with very high $(1 - 1/n)$ probability $f_e > 21\log n$ for *all* $e \in E$. Similarly, by Chernoff's bound stating that for $a \geq (2/3)E[f_e]$ we have $Pr[f_e < E[f_e] + a] < e^{-2E[f_e]/27}$ we have

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

So again, by union bound, we have that with very high $(1 - 1/n)$ probability $f_e < 70\log n$ for *all* $e \in E$. ∎

Recall that P_i is a shortest path from s to t in the graph $G \setminus e_i$, and an *interval* is a subpath of some P_i consisting of $n^{1-\alpha}$ consecutive vertices. We say that an interval of P_i *survives* if all the edges of this interval are present in some $G_j \in F_{e_i}$. We now show that all the n^2 possible intervals survive with very high probability as stated by the following lemma.

Lemma 2 *The probability that all the intervals survive is at least* $1 - 2/n$.

Proof: Consider some specific interval I of P_i, and some $G_j \in F_{e_i}$. The probability that all the $|I| - 1$ edges of I survived in G_j is precisely $(1 - n^{\alpha-1})^{|I|-1} \geq (1 - n^{\alpha-1})^{n^{1-\alpha}-1} > 1/e$. So the probability that I does *not* survive in any $G_j \in F_{e_i}$ is less than $(1 - 1/e)^{f_{e_i}}$. By Lemma 1, we can assume that $f_e > 21\log n$ and so $(1 - 1/e)^{f_{e_i}} < (1 - 1/e)^{21\log n} < (1/e)^{4\log n} < 1/n^4$. In particular, by union bound, we get that with probability $1 - 1/n^2$ all the possible n^2 intervals survive. ∎

After establishing that all intervals survive, we need to show that with very high probability every interval has at least one vertex in B. We choose B to be a random subset of $3n^\alpha \log n$ vertices of G and also require that $s, t \in B$.

Lemma 3 *With probability at least* $1 - 1/n$ *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 $1 - |B|/n = 1 - 3n^{\alpha-1}\log n$. So an entire interval does not belong to B with probability $(1 - 3n^{\alpha-1}\log n)^{n^{1-\alpha}} < 1/n^3$. There are at most n^2 intervals overall so by union bound we get that with probability $1 - 1/n$ every interval contains a vertex in B. ∎

C. Constructing the Graphs G^i_B.

Recall that G^i_B is a graph whose vertex set is B (that includes s and t), and the weight of the edge (u, v) is the length of the shortest u-to-v path in all $G_j \in F_{e_i}$. We can therefore construct G^i_B by computing for every G_j the $B \times B$ matrix B_j storing the distances between any two vertices in B. Once we have $B_1, \ldots, B_r$ every edge-weight in G^i_B can be computed by examining a single entry in every B_j such that $G_j \in F_{e_i}$. By Lemma 1 we know that $|F_{e_i}| < 70\log n$ and so G^i_B can be constructed in $O(|B|^2 \log n)$ time. So we can construct all G^i_Bs in $O(|B|^2 n \log n) = \tilde{O}(n^{2\alpha+1})$ time.

We are left with the problem of computing $B_1, \ldots, B_r$. We focus on computing a single B_j of the n-vertex graph G_j. For two vertices u, v let $c(u, v)$ denote the smallest number of edges on a shortest path from u to v and let $\delta(u, v)$ denote the distance from u to v.

Lemma 4 ([29]) *Given an n-vertex graph, the Yuster-Zwick algorithm constructs in $\tilde{O}(Mn^\omega)$ time, an $n \times n$ matrix D with the following properties: For any pair of vertices i, j there exists a vertex k on a shortest path realizing $c(i, j)$ so that $D[i, k] = \delta(i, k)$, $D[k, j] = \delta(k, j)$, and $D[i, k] + D[k, j] = \delta(i, j)$.*

We first run the Yuster-Zwick algorithm on G_j constructing the matrix D as in Lemma 4. Consider the sub-matrix D_1 of D which consists of taking only the rows that correspond to B. Let D_2 be the sub-matrix of D that consists only of the columns that correspond to B. It is easy to see that B_j is exactly the distance product $D_1 \star D_2$ (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 $D_1 \star D_2$ we use the following result, first stated by Alon et al. [31], following a related idea of Yuval [32].

Lemma 5 ([31]) *Let A be an $n^r \times n^s$ matrix and let B be an $n^s \times n^t$ 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, where $\omega(r, s, t)$ is the matrix multiplication exponent of multiplying an $n^r \times n^s$ matrix with an $n^s \times n^t$ matrix.*

We would like to compute $D_1 \star D_2$ using the bounds of Lemma 5. The problem is that the elements of D_1 and D_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_1 and D_2 whose absolute value is less than $Mn^{1-\alpha}$, the rest of the elements can be replaced by $+\infty$. This idea, together with the above lemmas, give the following.

Lemma 6 *Every matrix B_j can be computed in time $\tilde{O}(Mn^{\omega} + Mn^{1-\alpha}n^{\omega(\alpha,1,\alpha)})$.*

Proof: We need to show that any element of D_1 and D_2 whose absolute value is greater than $Mn^{1-\alpha}$ can be thought of as $+\infty$. To see this, consider a shortest path between two vertices $i, j \in B$. Recall that we assume every *interval* of $n^{1-\alpha}$ vertices has at least one vertex in B. Therefore, we are only interested in the i-to-j shortest path if the number of its edges $c(i,j) \leq n^{1-\alpha}$. In this case, by Lemma 4, there must be some $k \leq n$ such that $c(i,k) \leq n^{1-\alpha}$, and $c(k,j) \leq n^{1-\alpha}$, and $D[i,k] + D[j,k]$ is the length of the i-to-j shortest path. Since the absolute value of every edge-length is bounded by M, we get that $D[i,k] \leq Mn^{1-\alpha}$ and $D[j,k] \leq Mn^{1-\alpha}$. So the corresponding entries in D_1 and D_2 are bounded by $Mn^{1-\alpha}$.

■

By Lemma 6, we can compute all the B_j matrices in total time $\tilde{O}(rMn^{\omega} + rMn^{1-\alpha}n^{\omega(\alpha,1,\alpha)})$. Recalling that $r = \tilde{O}(n^{1-\alpha})$, that $\alpha \leq 1$, that $\omega \geq 2$, and that $\omega(\alpha,1,\alpha) \leq \omega\alpha + 1 - \alpha$ (See, e.g., Huang and Pan [34]), we have that $\tilde{O}(rMn^{\omega} + rMn^{1-\alpha}n^{\omega(\alpha,1,\alpha)}) = \tilde{O}(Mn^{1-\alpha+\omega})$.

Corollary 2 *All the G_B^i graphs can be constructed in total time $\tilde{O}(Mn^{1-\alpha+\omega} + n^{2\alpha+1})$.*

D. Computing s-to-t Shortest Paths in G_B^i

Finally, we need to compute the shortest s-to-t path in every G_B^i. Notice that the edge-lengths of G_B^i may be positive or negative, but they are no longer bounded by M. We may run Goldberg's SSSP algorithm (which can handle large integral negative lengths) on every G_B^i. However, we would like to use the faster Dijkstra algorithm. This algorithm can handle unbounded lengths but can not handle negative lengths. So, in order to use Dijkstra we must reweigh the edges so they become nonnegative, without changing the shortest s-to-t path. We achieve this by using *feasible price functions* and *reduced lengths*.

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 popular feasible price function comes from single-source distances. Let x be new vertex added to G with an edge from x to every other vertex of G having weight 0. Let $d(v)$ denote the length of the shortest path from x to v in $G \cup \{x\}$. 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 $G \cup \{x\}$ (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 every G_B^i. The following lemma states that we can use the same price function $d(v)$ on all G_B^is.

Lemma 7 *The function $d(\cdot)$ is a feasible price function for every G_B^i.*

Proof: Consider an edge (u, v) in some graph G_B^i. We need to prove that $w_d((u,v)) \geq 0$. We know that $w(u,v)$ is the length of the shortest u-to-v path in all $G_j \in F_{e_i}$. In particular, $w(u,v)$ is the length of some (not necessarily shortest) u-to-v path P in G. Consider the x-to-v path in G that is composed of the shortest x-to-u path in G and the path P. The length of this x-to-v path is $d(u) + w(u,v)$ and therefore $d(v) \leq d(u) + w(u,v)$ so $w_d((u,v)) = w(u,v) + d(u) - d(v) \geq 0$.

■

Notice that we can compute $d(v)$ for every $v \in G$ in $\tilde{O}(Mn^{\omega})$ time using the Yuster-Zwick algorithm (in fact, if every vertex is reachable from s then we can just use $x = s$ and recall that we have already computed the SSSP from s in the beginning of the algorithm to identify $e_1, \ldots, e_k$). Running Dijkstra on G_B^i is done in $O(|B|^2)$ time so on all G_B^is in $O(n|B|^2) = \tilde{O}(n^{1+2\alpha})$ time.

Total time complexity: From the above description, the total time complexity of our replacement paths algorithms is

$$\tilde{O}(Mn^{\omega} + n^{3-\alpha} + Mn^{1-\alpha+\omega} + n^{1+2\alpha}).$$

Taking α to be $\omega/3$ gives total time complexity $\tilde{O}(Mn^{1+\frac{2}{3}\omega})$ thus proving Theorem 1.

III. Extensions to All Pairs Replacement Paths

In this section we describe the required changes to turn our replacement paths solution into an *all pairs* replacement paths solution. We begin with the case of avoiding a single edge. We present a distance sensitivity oracle that upon query (u, v, e) outputs the length of the shortest u-to-v path that avoids the edge e. We then show how to extend our distance sensitivity oracle to handle multiple failures. A query (u, v, S) to this oracle returns the length of the shortest u-to-v path that avoids all the edges in S.

A. Handling a Single Failure

We now describe a distance sensitivity oracle for *single* edge failure in the bounds of Theorem 2. We divide the algorithm from the previous section into a preprocessing stage and a query stage. In the preprocessing stage, without yet knowing what edge will fail, we construct the random subgraphs G_j. We also choose the random subset of vertices B and compute the $B \times B$ shortest path matrices B_j. In the query stage, knowing (u, v, e), we first identify the set F_e of all G_js that exclude e. We then add u and v to B and accordingly extend every matrix B_j such that $G_j \in F_e$. Using these extended B_j matrices we can construct the graph G_B^e. Finally, we use reduced lengths and run Dijkstra's algorithm on G_B^e (a graph with B vertices, where the weight of the edge (u, v) is the length of the shortest u-to-v path in all $G_j \in F_e$).

The preprocessing stage is done exactly as in the previous section so the graphs G_j are constructed in $\tilde{O}(n^{3-\alpha})$ time. In particular, Lemmas 1, 2, 3, and 6 still hold and so computing all B_j matrices can be done (as in Section II-C) in $\tilde{O}(Mn^{1-\alpha+\omega})$ time and $O(r \cdot |B|^2) = \tilde{O}(n^{1+\alpha})$ space.

In the query stage, computing the set F_e of all G_js that exclude e can be done in $O(r) = \tilde{O}(n^{1-\alpha})$ time by checking each of the G_js in constant time. We then need to modify every matrix B_j where $G_j \in F_e$. By Lemma 1, there are only $O(\log n)$ such B_j matrices as $|F_e| = O(\log n)$. Every such B_j currently holds the $B \times B$ distances in G_j and we would like to add to it the $\{u\} \times B$ and $B \times \{v\}$ distances. To do so, we recall that during the construction of B_j in the preprocessing stage, we got an $n \times n$ matrix D such that $D \star D$ is the all pairs distance matrix of G_j. We compute the $\{u\} \times B$ and the $B \times \{v\}$ distances from D. Each distance is computed in $O(n)$ time for a total of $O(|B|n) = \tilde{O}(n^{1+\alpha})$ time. The next step is to construct G_B^e. We compute each of its $|B|^2$ edge-lengths by looking at a single entry in $|F_e|$ matrices B_j for a total of $O(|B|^2|F_e|) = \tilde{O}(n^{2\alpha})$ time. Finally, a single execution of Dijkstra is done on the graph G_B^e using reduced lengths in $O(|B|^2)$ time. The overall running time of the query stage is, therefore, $\tilde{O}(n^{1+\alpha})$.

B. Handling Multiple Failures

We would like to extend our distance sensitivity oracle from avoiding one edge to avoiding any fixed number of $f \geq 2$ edges. A query (u, v, S) now asks for the shortest u-to-v path that avoids all edges in the set S (where $f = |S|$). The oracle is essentially the same as the one we just described for avoiding a single edge. We outline the differences leading to the bounds of Corollary 1.

We begin with the randomized analysis. We now generate the random subgraphs $G_1, \ldots, G_r$ by removing every edge with probability $n^{(\alpha-1)/f}$. This time we choose $r = 42fn^{1-\alpha} \log n$. Let F_S denote the set of graphs G_j that do not contain *any* edges in S, and let $f_S = |F_S|$:

- Lemma 1 should now state $21f \log n < f_S < 70f \log n$. The expectation $E[f_S] = r(n^{(\alpha-1)/f})^f = rn^{\alpha-1}$ just as before and the proof of Lemma 1 holds. We also change the length of an *interval* to be $n^{(1-\alpha)/f}$. An interval is now said to *survive* if all its edges appear in some $G_j \in F_S$.
- In Lemma 2, we now get that an interval I survives in G_j with probability $(1 - n^{(\alpha-1)/f})^{|I|-1} > 1/e$. So the probability that I does *not* survive in any $G_j \in F_S$ is less than $(1 - 1/e)^{f_S} < 1/n^{6f}$. Notice that the total number of possible intervals is no longer n^2, it is now n^{3+f} since there are n^2 pairs of vertices, each pair is responsible for n^f replacement paths, and each replacement path has up to n intervals. Lemma 2 holds because, by union bound, we get that with probability $1 - n^{3+f}/n^{6f} > 1 - 1/n^2$ all the possible n^{3+f} intervals survive.
- Finally, for Lemma 3, we change $|B|$ to be $3fn^{1+(\alpha-1)/f} \log n$. A specific vertex v does not belong to B with probability $1 - |B|/n = 1 - 3fn^{(\alpha-1)/f} \log n$. So an entire interval does not belong to B with probability $(1 - 3fn^{(\alpha-1)/f} \log n)^{n^{(1-\alpha)/f}} < 1/n^{3f}$. There are at most n^{3+f} intervals overall so by union bound we get that with probability $1 - n^{3+f}/n^{3f} > 1 - 1/n$ every interval contains a vertex in B. Lemma 3 follows.

In the preprocessing stage, when computing B_j (Section II-C), since an interval length is now $n^{(1-\alpha)/f}$ we now only need to consider elements of D (See Lemma 4) whose absolute value is bounded by $Mn^{(1-\alpha)/f}$. This fact is used to bound $L = Mn^{(1-\alpha)/f}$ in Lemma 5. In the same lemma, since $|B|$ is now $3fn^{1+(\alpha-1)/f} \log n$, the distance product of a $B \times n$ matrix and an $n \times B$ matrix is computed in $\tilde{O}(Ln^{\omega(r,s,t)})$ time with $\omega(r, s, t) = \omega(1 + (\alpha - 1)/f, 1, 1 + (\alpha - 1)/f)$. Lemma 6 should now state that constructing a single B_j requires time $\tilde{O}(Mn^{\omega} + Mn^{(1-\alpha)/f} n^{\omega(1+(\alpha-1)/f, 1, 1+(\alpha-1)/f)})$. Multiplying this by r (recalling that $r = \tilde{O}(n^{1-\alpha})$, that $\alpha \leq 1$, that $\omega \geq 2$, and that $\omega(x, 1, x) \leq \omega x + 1 - x$) we get that the total time to construct all B_js is $\tilde{O}(Mn^{1-\alpha+\omega})$. The total space required for all B_js is $O(r \cdot |B|^2) = \tilde{O}(f^3 n^{3-\alpha+(2\alpha-2)/f})$.

In the query stage, computing the set F_S can be done in $O(fr) = \tilde{O}(n^{1-\alpha})$ time by checking each of the G_js in $O(f)$ time. We then need to modify every matrix B_j where $G_j \in F_S$. There are only $O(f \log n)$ such B_j matrices as $|F_S| = O(f \log n)$. Adding the $\{u\} \times B$ and $B \times \{v\}$ distances to all B_js is done as in Section III-A in $O(|F_S| \cdot |B|n) = \tilde{O}(f^2 n^{2-(1-\alpha)/f})$ time. Finally, we construct G_B^S: Its set of vertices is B, and the weight of the edge (u, v) is the length of the shortest u-to-v path in all $G_j \in F_S$. The construction of G_B^S and the execution of Dijkstra is done similarly to Section II in $O(|B|^2|F_S|) = \tilde{O}(f^3 n^{2-2(1-\alpha)/f})$ time. The total query time is thus $\tilde{O}(fn^{1-\alpha} + f^2 n^{2-(1-\alpha)/f} + f^3 n^{2-2(1-\alpha)/f}) = \tilde{O}(n^{2-(1-\alpha)/f})$. Corollary 1 follows.

IV. Concluding Remarks

We have presented an algorithm for the replacement path problem in weighted directed graphs. Our algorithm uses fast matrix multiplication and runs in $\tilde{O}(Mn^{1+\frac{2}{3}\omega})$ time on an n-node graph with weights in $\{-M, \ldots, M\}$. We also showed how to extend this solution to an $\tilde{O}(Mn^{1-\alpha+\omega})$-time constructible oracle that upon query (u, v, S) returns the length of the shortest u-to-v path that avoids all the edges in the set S in $\tilde{O}(n^{2-(1-\alpha)/f})$ time.

Both of these solutions can be made to work in the case of avoiding vertices rather than edges. To achieve that, each random subgraph G_j is now obtained from G by removing every *vertex* (and its adjacent edges) with probability $n^{\alpha-1}$. Notice that the vertices of B might also be removed. However, by a similar argument to Lemma 2, *all* the vertices of an interval survive in some G_j. In particular, by Lemma 3, some vertex of B is contained in the interval and is thus not removed.

The main open question is whether we can solve the replacement paths problem in subcubic time on directed graphs with unbounded (i.e, independent of M) weights.

References

[1] N. Nisan and A. Ronen, "Algorithmic mechanism design," *Games and Economic Behavior*, vol. 35, pp. 166–196, 2001.

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

[3] D. Eppstein, "Finding the k shortest paths," *SIAM J. Comput.*, vol. 28, pp. 652–673, 1999.

[4] Z. Gotthilf and M. Lewenstein, "Improved algorithms for the k simple shortest paths and the replacement paths problems," *Information Processing Letters*, vol. 109, no. 7, pp. 352–355, 2009.

[5] J. Yen, "Finding the k shortest loopless paths in a network," *Management Science*, vol. 17, pp. 712–716, 1971.

[6] E. Lawler, "A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem," *Management Science*, vol. 18, pp. 401–405, 1972.

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

[8] D. Karger, D. Koller, and S. J. Phillips, "Finding the hidden path: Time bounds for all-pairs shortest paths," *SIAM J. Comput.*, vol. 22, no. 6, pp. 1199–1217, 1993.

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

[10] E. Nardelli, G. Proietti, and P. Widmayer, "Finding the most vital node of a shortest path," *Theoretical Computer Science*, vol. 296, no. 1, pp. 167–177, 2003.

[11] ——, "A faster computation of the most vital edge of a shortest path," *Information Processing Letters*, vol. 79, no. 2, pp. 81–85, 2001.

[12] M. Thorup, "Undirected single-source shortest paths with positive integer weights in linear time," *Journal of the ACM*, vol. 46, pp. 362–394, 1999.

[13] L. Roditty and U. Zwick, "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)*, 2005, pp. 249–260.

[14] Y. Emek, D. Peleg, and L. Roditty, "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)*, 2008, pp. 428–435.

[15] P. Klein, S. Mozes, and O. Weimann, "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)*, 2009, pp. 236–245.

[16] C. Wulff-Nilsen, "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)*, 2010, pp. 756–765.

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

[18] A. Bernstein, "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)*, 2010, pp. 742–755.

[19] C. Demetrescu, M. Thorup, R. Chowdhury, and V. Ramachandran, "Oracles for distances avoiding a failed node or link," *SIAM J. Comput.*, vol. 37, no. 5, pp. 1299–1318, 2008.

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

[21] ——, "A nearly optimal oracle for avoiding failed vertices and edges," in *Proc. of the 41st ACM Symposium on Theory of Computing (STOC)*, 2009, pp. 101–110.

[22] R. Duan and S. Pettie, "Dual-failure distance and connectivity oracles," in *Proc. of the 20th ACM-SIAM Symposium On Discrete Algorithms (SODA)*, 2009, pp. 506–515.

[23] S. Chechick, M. Langberg, D. Peleg, and L. Roditty, "f-sensitivity distance oracles," *Unpublished manuscript*, 2009.

[24] C. Demetrescu and G. Italiano, "Experimental analysis of dynamic all pairs shortest path algorithms," in *Proc. of the 15th ACM-SIAM Symposium On Discrete Algorithms (SODA)*, 2004, pp. 362–371.

[25] M. Thorup, "Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles," in *Proc. of the 9th Scandinavian Workshop on Algorithm Theory (SWAT)*, 2004, pp. 384–396.

[26] L. Roditty and U. Zwick, "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)*, 2004, pp. 184–191.

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

[28] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *Journal of Symbolic Computation*, vol. 9, pp. 251–280, 1990.

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

[30] A. Goldberg, "Scaling algorithms for the shortest paths problem," in *Proc. of the 4th ACM-SIAM Symposium On Discrete Algorithms (SODA)*, 1993, pp. 222–231.

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

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

[33] N. Alon and J. Spencer, *The probabilistic method*, 2nd ed. Wiley-Interscience, 2000.

[34] X. Huang and V. Pan, "Fast rectangular matrix multiplications and applications," *Journal of Complexity*, vol. 14, pp. 257–299, 1998.