Incremental Distance Products via Faulty Shortest Paths

Oren Weimann[*]
University of Haifa, Israel

Raphael Yuster[†]
University of Haifa, Israel.

Abstract

Given a constant number d of $n \times n$ matrices with total m non-infinity entries, we show how to construct in (essentially optimal) $\tilde{O}(mn + n^2)$ time a data structure that can compute in (essentially optimal) $\tilde{O}(n^2)$ time the distance product of these matrices after incrementing the value (possibly to infinity) of a constant number k of entries.

Our result is obtained by designing an oracle for *single source replacement paths* that is suited for short distances: Given a graph $G = (V, E)$, a source vertex s, and a shortest paths tree T of depth d rooted at s, the oracle can be constructed in $\tilde{O}(md^k)$ time for any constant k. Then, given an arbitrary set S of at most $k = O(1)$ edges and an arbitrary vertex t, the oracle in $\tilde{O}(1)$ time either reports the length of the shortest s-to-t path in $(V, E \setminus S)$ or otherwise reports that any such shortest path must use more than d edges.

1 Introduction

The distance product $A \otimes B$ of two $n \times n$ matrices A, B with entries in $\mathbb{R} \cup \infty$ is the $n \times n$ matrix C with $C[i,j] = \min_{\ell=1}^{n} A[i,\ell] + B[\ell,j]$. It is well known that computing the distance product is equivalent to solving All Pairs Shortest Paths (APSP). In particular, the distance product of two $n \times n$ matrices with m non-infinity entries is equivalent to APSP in graphs with n vertices and m edges. Both problems can be solved in $\tilde{O}(nm + n^2)$ time and this is optimal under the popular conjecture that there is no $O((mn)^{1-\epsilon})$ time algorithm for APSP.

In this paper, we consider the problem of maintaining a distance product of a constant number of matrices subject to a constant number of entries being incremented. Namely, let k and d be two constants. We are given $n \times n$ matrices $A_1, \ldots, A_d$ with entries in $\mathbb{R} \cup \infty$ such that a total of m entries are non-infinity. A *location increment* is a quadruple (ℓ, i, j, δ) corresponding to entry $A_\ell[i,j]$ being increased by δ. Given a set S of k location increments, let $C(S)$ be the distance product $A_1 \otimes \cdots \otimes A_d$ after applying these k location increments. A naïve solution is, given S, to compute $C(S)$ from scratch in $\tilde{O}(nm + n^2)$ time. Instead, we prove that after an $\tilde{O}(nm + n^2)$ time preprocessing stage, given S we can compute $C(S)$ in $\tilde{O}(n^2)$ time. We achieve this by designing an efficient solution to the *single source replacement paths* problem that is suited for short distances.

1.1 Single Source Replacement Paths

Given two vertices s and t and an edge e, a replacement path $P_{s,t,e}$ is a shortest s-to-t path that avoids e. In the *Replacement Paths* (RP) problem, we are given a graph G and a shortest path P between two vertices s and t. The goal is to return the length of a replacement path $P_{s,t,e}$ for every edge e of P. Note that the size of the RP output is $\Theta(n)$. In the *Single Source Replacement*

[*]Supported partially by ISF grant 592/17
[†]Supported partially by ISF grant 1082/16

Paths (SSRP) problem, we are given a graph G and a shortest path tree T rooted at s. The goal is to return the length of a replacement path $P_{s,t,e}$ for every vertex t of G and every edge e of T. Note that the size of the SSRP output is $\Theta(n^2)$.

Arbitrary edge weights. The naïve solution to both RP and SSRP is to remove each edge e (in RP $e \in P$ and in SSRP $e \in T$) one at a time, and compute a shortest path tree each time. For general directed graphs with n vertices, m edges, and arbitrary edge weights, this can easily be done in $O(mn + n^2 \lg n)$ time. The fastest known algorithm is $O(mn + n^2 \lg\lg n)$ [13], and the existence of an $O((mn)^{1-\epsilon})$ algorithm seems unlikely: First, Hershberger et al. [15] showed that in the path-comparison model of computation of Karger et al. [17] SSRP requires $\Omega(mn)$ comparisons. Second, Vassilevska Williams and Williams [24] showed that a subcubic $O(n^{3-\epsilon})$ time RP algorithm implies a subcubic time algorithm for All-Pairs Shortest Paths (APSP). Therefore, in directed dense graphs with arbitrary edge weights, the cubic time complexity of both RP and SSRP seems unavoidable.

Unweighted graphs and undirected graphs. For unweighted directed graphs, Roditty and Zwick [21] showed that RP can be solved in $\tilde{O}(m\sqrt{n})$. Their algorithm was randomized and was recently derantomized by Alon Chechik and Cohen [1]. For undirected graphs, Malik et al. [18] and Nardelli et al. [19, 20] showed that RP can be solved in near linear $\tilde{O}(m)$ time. Finally, for undirected unweighted graphs, Chechik and Cohen [8] presented a randomized $\tilde{O}(m\sqrt{n} + n^2)$ time algorithm for SSRP along with a matching conditional lower bound (based on the hardness of Boolean Matrix Multiplication).

Bounded integral weights. When the edge weights are integers in $[-M, M]$, there are faster algorithms for both RP and SSRP that are based on fast matrix multiplication. Weimann and Yuster [22] described a randomized $\tilde{O}(Mn^{1+\frac{2}{3}\omega})$ algorithm for RP which was later improved by Vassilevska Williams [25] to $\tilde{O}(Mn^{\omega})$. As for SSRP, Grandoni and Vassilevska Williams [14] presented an $\tilde{O}(Mn^{\omega})$ time randomized algorithm for positive integral edge weights in $[1, M]$, and an $\tilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$ time randomized algorithm for positive and negative integral weights in $[-M, M]$. Observe that with positive weights, the same $\tilde{O}(Mn^{\omega})$ bound is known for both RP and SSRP. This is also the best known bound for Single-Source Shortest Paths (SSSP) [26]. With negative weights however, there is currently a gap between RP and SSRP with the latter having an $\tilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$ bound matching the best known bound for All-Pairs Shortest Paths (APSP) [27]. Grandoni and Vassilevska Williams [14] conjectured that the reason for this gap is that the SSRP problem with negative weights is as hard as APSP.

Distance sensitivity oracles. In the more general version of the problem, we wish to design a data structure (a *distance sensitivity oracle*) that is not restricted to a single source. That is, can return the length of $P_{s,t,e}$ given any two vertices s, t and any edge e. In the most general setting, such an oracle could handle multiple failures (i.e. given vertices s, t and a set of at most k edges return the shortest s-to-t path that avoids all edges in the set). Distance sensitivity oracles have been extensively studied. In particular, Grandoni and Vassilevska Williams [14] showed a reduction from (single-fault) distance sensitivity oracles to the SSRP problem.

A naïve distance sensitivity oracle is to precompute and store for every vertex v and for every set of edges F with $|F| \leq k$, the distances from v to all other vertices in $G \setminus F$. This requires $O(n^2 \cdot m^k)$ space and $\tilde{O}(k)$ query. The first nontrivial distance sensitive oracle for a single edge failure ($k = 1$) in directed graphs was given by Demetrescu et al. [11] who obtained space $O(n^2 \log n)$ and query time $O(\log n)$. This was extended in [10] to a single edge *or vertex* failure with space $O(n^2 \log n)$, query time $O(1)$, and preprocessing time $\tilde{O}(mn^2)$. Later, Karger and Bernstein improved the preprocessing time (while keeping the space and query time unchanged), first to $O(n^2\sqrt{m})$ in [3] and then to $\tilde{O}(mn)$ in [4]. For two failures, an oracle of size $O(n^2 \log^3 n)$ with query time $O(\log n)$ and polynomial preprocessing time was given Duan and Pettie [12]. For k failures, an oracle of size $O(n^2 \cdot n^k)$ with query time $O(k^2)$ was given by Chechik et al. [9].

Improved oracles are known for bounded integral edge weights [14, 23] (with faster preprocessing time at the cost of a polynomial query time), and when we are willing to settle for approximate distances [2,5,6,7,9].

1.2 Our results

We consider the SSRP problem in directed graphs with arbitrary (positive and negative) edge weights. We assume that the graph contains no negative weight cycles.

- In Section 2 we refine the $\tilde{\Theta}(mn)$ known upper and lower bounds for SSRP to be parametrized by the (unweighted) depth d of the shortest path tree T. Namely, if we only wish to compute replacement paths to vertices of depth at most d then (1) there is a deterministic $\tilde{O}(md)$ time algorithm for SSRP, and (2) an $O((md)^{1-\epsilon})$ time SSRP algorithm for any d is unlikely as it implies an $O((mn)^{1-\epsilon})$ time algorithm for SSSP with negative weights. The lower bound is quite trivial and the upper bound could be easily obtained by using [10, Corollary 4]. We give a slightly different solution than [10] that we can then extend to multiple faults.

- In Section 3 we show how to handle SSRP with k faults. Namely, how to compute for every t and every set of k edges the weight of a shortest s-to-t path that avoids all the k edges in the set. We give a deterministic $O((m+n\log n)(d\log n)^k)$ time algorithm for the case where we know that all replacement paths contain at most d vertices. For instance, the family of DAG's (directed acyclic graphs) of depth d has this property. In fact, we give a single source distance sensitivity oracle with preprocessing time $O((m+n\log n)(d\log n)^k)$ and query time $O(k\log^k n)$. Given an arbitrary set of at most k faulty edges and an arbitrary vertex t the oracle reports the s-to-t distance that avoids all the k edges or otherwise reports that such path must use more than d edges. We use our oracle for the case when $k = O(1)$ and then the preprocessing is $\tilde{O}(md^k)$ and the query time is $\tilde{O}(1)$.

- In Section 4 we use our oracle to solve the problem of maintaining a distance product. Given $n \times n$ matrices $A_1, \ldots, A_d$ with entries in $\mathbb{R} \cup \infty$ where m entries are non-infinity, we show how to construct in $O((nm + n^2\log n)(d\log n)^k)$ time a data structure that can compute in $O(n^2 2^k k \log^k n)$ time the distance product $C = A_1 \otimes \cdots \otimes A_d$ after k entries are changed (to a larger, possibly infinite value). This is interesting when both k and d are constants as then: (1) The construction time is $\tilde{O}(mn + n^2)$ and is optimal under the conjecture that there is no $O((mn)^{1-\epsilon})$ time algorithm for APSP with real edge lengths on graphs with n vertices and m edges, and (2) The query time is $\tilde{O}(n^2)$ and is optimal up to logarithmic factors since even for $d = 3$ a single fault can change all the n^2 entries of C.

2 SSRP with a Single Fault

We assume that the (unweighted) depth of the shortest paths tree T rooted at s is d (otherwise, we consider the first d levels of T). We can compute T in $O(md)$ time by running d iterations of Bellman-Ford (i.e. stopping at the iteration when no edges are relaxed). We then use the standard trick of Johnson [16] to reweigh the edges of the graph to become nonnegative. Namely, we add to the weight of edge (u, v) the s-to-u distance and subtract the s-to-v distance. This reweighing does not change the identity of shortest paths nor of replacement paths (and their original length can be recovered from their new length in constant time). Moreover, the new weights of all edges in T are zero.

After reweighing, we partition the edges of T into d levels according to their (unweighted) depth. Then, for each level ℓ we compute in $O(m \log n + n \log^2 n)$ time the replacement paths $P_{s,t,e}$ of all vertices t of G and all edges e of level ℓ. To achieve this, observe that we can restrict the paths $P_{s,t,e}$ to visit at most one edge of T that is of level ℓ. Otherwise, if $P_{s,t,e}$ first visits edge (u, v) and later edge (u', v') (both of level ℓ) then there is an alternative path to u' that is of length zero and does not visit any level-ℓ edge (the s-to-u' path in T).

To compute all $P_{s,t,e}$ of level-ℓ edges e, we therefore apply the following procedure: We consider the level-ℓ edges as distinct binary strings with $O(\log n)$ bits and proceed in $O(\log n)$ iterations. In the i'th iteration we run Dijkstra's algorithm from s on the graph obtained from G by removing all level-ℓ edges whose i'th bit is 1. We then do the same with the graph obtained from G by removing all level-ℓ edges whose i'th bit is 0. Now, suppose that a certain $P_{s,t,e}$ uses the level-ℓ edge e' (and no other level-ℓ edges). Since e and e' differ in at least one bit, we know that in one of the Dijkstra runs in which e was removed e' survived. So $P_{s,t,e}$ was captured then.

There are d levels and for each of them we run Dijkstra's algorithm $O(\log n)$ times so the overall time is $\tilde{O}(md)$. We summarize our result in the following theorem.

Theorem 1. *Given a directed graph G with m arbitrary weighted edges and a shortest paths tree of depth d rooted at s, there is a deterministic $\tilde{O}(md)$ time algorithm for the SSRP problem from vertex s.*

Lower bound. We now show that in real-weighted graphs, for any d, an algorithm that finds all replacement paths from s to every vertex of depth at most d in $O((md)^{1-\epsilon})$ time[1] implies an algorithm for Single-Source Shortest Paths (SSSP) with negative weights in $O((mn)^{1-\epsilon})$ time.

Clearly, finding the replacement paths to all shortest paths of at most d edges is at least as hard as finding all shortest paths of at most d edges. If the latter could be solved in $O((md)^{1-\epsilon})$ time then we could apply it n/d times to solve SSSP with negative weights in $O((md)^{1-\epsilon} \cdot (n/d)) = O(m^{1-\epsilon} n)$ time. After each iteration, for every vertex v that was reached we add a directed edge (s, v) whose length is the s-to-v distance (if there was already an (s, v) edge then we just change its length). The new graph still has $O(m)$ edges but has the property that the number of edges in each shortest path decreases by $d - 1$, so after $O(n/d)$ iterations we are done.

3 SSRP with Multiple Faults

In this section we give a single source distance sensitivity oracle with preprocessing time $O((m + n \log n)(d \log n)^k)$ and query time $O(k \log^k n)$. Given an arbitrary set S of at most k faulty edges and an arbitrary vertex t a query reports the s-to-t distance that avoids all the k edges or otherwise reports that any such path must use more than d edges.

The data structure. Let $G = (V, E)$ denote the input graph and s denote the source vertex. Suppose that $|V| = n$, $|E| = m$ and each $e \in E$ is uniquely represented by a binary string

[1]In fact, it is enough to assume $O(m^{1-\epsilon} d)$.

$(e_1, \ldots, e_r)$ where $r = \lceil \log m \rceil$. Our data structure is represented by a perfect q-ary tree Q with $k+1$ levels where $q = 2rd$. Shortest paths information is held only in the leaves of Q which are at level k and internal vertices are used in order to traverse the tree to the correct leaf at query time. Each vertex of Q is associated with a label Q_F for some subset $F \subseteq E$ of edges. The label of the root of Q is Q_E. Let $G[F]$ denote the subgraph of G consisting of the edges F and let T_F denote a shortest path tree of $G[F]$ rooted at s. Let A_F be an array indexed by V where $A_F[v]$ is the level of vertex v in T_F (s has level zero). In every internal vertex of Q with label Q_F we store the entire array A_F.

We next show how to construct and label the children of an internal vertex Q_F given its corresponding array A_F and its shortest paths tree T_F. Since Q_F has $2rd$ children, we consider each child $Q_{F'}$ as a triple (ℓ, j, b) where $0 \leq \ell < d$, $1 \leq j \leq r$ and $b \in \{0, 1\}$. $F' \subset F$ is obtained from F by removing all edges $e = (u, v)$ where u is in level ℓ of T_F (i.e. $A_F[u] = \ell$ and $A_F[v] = \ell + 1$) and furthermore $e_j = b$ (i.e. the jth bit in the binary representation of e is b). We also remove any edge of F whose head is in level larger than d (we do not care about such edges as recall that we are only interested in shortest paths trees having depth at most d). This uniquely defines F'. We say that the child $Q_{F'}$ is the (ℓ, j, b)-child of Q_F.

Finally, we define the information kept in the leaves of Q. Consider some leaf labeled Q_F. Recall that we may assume that all our edges have nonnegative weight as we have initially used Johnson's reweighing trick on the input graph, as described in the previous section. The information kept in the leaf Q_F is just the result of a Dijkstra run from s in $G[F]$. Recall also that we can recover the original lengths before the reweighing in constant time per length.

The query. First observe that if a distance sensitivity oracle can handle exactly k faults, then it can be used to handle at most k faults as well (one simple way to see this is by adding "dummy edges" and using them to complement queries with fewer than k edges to queries with precisely k edges). We therefore describe how to use the tree Q to answer a query (S, t) where $S \subset E$ is an arbitrary set of exactly k edges and $t \in V$ is an arbitrary vertex, and we seek the length of a shortest s-to-t path in the graph $G[E \setminus S]$. By considering the root Q_E of Q and the array A_E we locate an edge $e = (u, v) \in S$ with minimum $A_E[u]$ among all heads of the edges in S (if there are ties, we pick e arbitrarily). Suppose $A_E[u] = \ell$ and $e = (e_1, \ldots, e_r)$. We recursively call the r children (ℓ, j, e_j) for $j = 1, \ldots, r$ providing each recursive call the query $(S \setminus e, t)$. Each of the r recursive calls will provide an answer (a candidate answer for the query), we pick the smallest and return this as the query's answer. More generally, when we are presently at vertex Q_F of level p in Q, we have a partial query of the form (S', t) where $S' \subseteq S$, $|S'| = k - p$. If $p = k$ we are at a leaf of Q so we just return in $O(1)$ time the length of the shortest s-to-t path in $G[F]$ that is stored in Q_F. Otherwise, $p < k$ and Q_F is an internal vertex of Q, so we recurse as described above.

Runtime. Running d iterations of Bellman-Ford and using Johnson's trick is done initially and requires $O(md)$ time. The information stored at each vertex of Q is obtained by a single Dijkstra run on a graph with n vertices and at most m edges. As the number of vertices of Q is $O((d \log n)^k)$, the total time to construct Q is $O((m + n \log n)(d \log n)^k)$.

A query visits $O(\log^k n)$ vertices of Q. In each visit, the edge e of the current set S' with minimum head level is located in $O(k)$ time as $|S'| \leq k$. At a leaf Q_F we spend only $O(1)$ time to get the length of the shortest s-to-t path in $G[F]$. Thus, a query takes $O(k \log^k n)$ time.

Correctness. The query correctness follows from the following claim: Suppose we are presently at vertex Q_F at level p of Q and our partial query is (S', t). Then, the result returned by this vertex to its caller is the length of a shortest s-to-t path in $G[F]$ that uses at most

d edges and avoids the edges in S'. We prove the claim by induction on $k-p$. For the base case $k-p=0$ so Q_F is a leaf and $S'=\emptyset$. Then indeed the information stored at Q_F contains the length of the shortest s-to-t path in $G[F]$ and this length is returned. Assuming the claim holds for level $k \geq p+1>0$ we prove it holds for level p. So we are presently at an internal vertex Q_F at level p of Q and our partial query is (S',t) with $|S'|=k-p$. Let $e=(u,v) \in S'$ with $A_F[u]=\ell$ minimum among all heads of the elements of S'. As shown in the previous section, we may restrict to shortest s-to-t paths in $G[F]$ which avoid S' and contain at most one edge whose head is at level ℓ of T_F. Suppose that a certain such path P uses an edge $e'=(u',v')$ whose head is at level ℓ of T_F (and P uses no other edge whose head is at level ℓ of T_F). Suppose that the j'th bit in the binary representation of e differs from the j'th bit in the binary representation of e', that is $e'_j = 1-e_j$. One of the children we recursively call is the (ℓ, j, e_j) child of Q_F, labeled Q'_F and provide it with the partial query $(S' \setminus e, t)$. Observe that all the edges of P survived in F'. By the induction hypothesis, this child will return the length of a shortest s-to-t path in $G[F']$ which avoids the edges in $S' \setminus e$, namely it will return the length of P. All other children we call return values that are at least as large as the length of P as in the other calls, P might not have survived, nor any other shortest s-to-t path of $G[F]$ that avoids S'. As we are returning the minimum over all calls, we will correctly return the length of P.
We summarize our result in the following theorem.

Theorem 2. *Let k be a fixed positive integer. Given a directed graph G with m arbitrary weighted edges and a source vertex s, there is a deterministic distance sensitivity oracle which given a query (S,t) where S is an arbitrary subset of k edges and t is an arbitrary vertex, returns in $O(k \log^k n)$ time (so $\tilde{O}(1)$ for constant k) the shortest s-to-t path that avoids S or else reports that any such path uses more than d edges. The time to construct the oracle is $O((m+n\log n)(d \log n)^k)$ (so $\tilde{O}(md^k)$ for constant k).*

4 Distance Product with Faults

Given $n \times n$ matrices $A_1, \ldots, A_d$ with entries in $\mathbb{R} \cup \infty$, we show how to construct in $O((nm + n^2 \log n)(d \log n)^k)$ time a data structure that can compute in $O(n^2 2^k k \log^k n)$ time the distance product $C = A_1 \otimes \cdots \otimes A_d$ after k entries are changed (each to a larger, possibly infinite value).

Our data structure uses the distance sensitivity oracle from the previous section, but requires an additional lemma (of independent interest) showing that any distance sensitivity oracle that handles k faults with query time q can be used to obtain an *edge increase distance oracle* with the same preprocessing time and query time $O(2^k \cdot q)$. While we state this lemma for single source distance sensitivity oracles, it is clear that it holds for multiple source distance sensitivity oracles as well. Formally, a *single source k edge-increase distance oracle* for a graph G and a source vertex s is a data structure that is given queries of the form (S,t) where t is an arbitrary vertex and S is a set of k pairs $(e_1,\delta_1), \ldots, (e_k,\delta_k)$ where e_i is an edge and δ_i is a positive real, possibly infinity. The query's goal is to return the shortest s-to-t path in G after the weights of e_i increased by δ_i. Observe that a standard single source distance sensitivity oracle is a special case, where all the δ_i are infinity.

Lemma 3. *For a graph G, a source vertex s, and a number k, any k-fault single source distance sensitivity oracle with query time q implies a k edge-increase single source distance oracle for G and s using the same data structure and whose query time is $O(2^k \cdot q)$.*

Proof. We show how to process a query (S,t) where now S is a set of k pairs $(e_1,\delta_1), \ldots, (e_k,\delta_k)$ as above. Let G' be the same as G but after the weight of each e_i is increased by δ_i. Consider a shortest s-to-t path P in G'. Let $F=\{e_1, \ldots, e_k\}$ and let $F' = F \cap P$ denote the set of edges of F

used in P. Observe that P is also a shortest path in the graph $G[E \setminus (F \setminus F')]$ (the original G after the edges of $F \setminus F'$ are removed). Therefore, if we query the distance sensitivity oracle with query $(F \setminus F', t)$ it will return the weight w of P in $G[E \setminus (F \setminus F')]$. Then the weight of P in G' is just $w + \sum_{e_i \in F'} \delta_i$. Of course, we do not know what is F' so we proceed as follows. For each possible subset $F' \subseteq F$, query the distance sensitivity oracle with the query $(F \setminus F', t)$. Letting w_F be its answer, set $w^*_F = w_F + \sum_{e_i \in F'} \delta_i$. Notice that if $F' \neq F \cap P$ then w^*_F is at least as large as the weight of P in G'. Taking the minimum of all the w^*_F therefore correctly returns the weight of P in G'. As we have queried the distance sensitivity oracle 2^k times, the query time of the k edge-increase single source distance oracle is $O(2^k \cdot q)$. ■

The data structure. Given the matrices $A_1, \ldots, A_d$ we construct an acyclic $(d+1)$-layered graph G as follows. The vertex set of G consists of $d+1$ parts denoted $V_1, \ldots, V_{d+1}$ where $V_\ell = \{v_{\ell,1}, \ldots, v_{\ell,n}\}$. There are edges only from a part V_ℓ to the next part $V_{\ell+1}$. More precisely, for each non-infinity entry of A_ℓ, say, $A_\ell[i,j]$ we add an edge from $v_{\ell,i}$ to $v_{\ell+1,j}$ with weight $A_\ell[i,j]$. Observe that any path in G contains at most d edges and any path from a vertex in V_1 to a vertex in V_{d+1} contains precisely d edges. Furthermore, G has $n(d+1) = O(n)$ vertices and m edges. If $C = A_1 \otimes \cdots \otimes A_d$ then $C[i,j]$ is the weight of a shortest path in G from $v_{1,i}$ to $v_{d+1,j}$. Similarly, $C(S)[i,j]$ is the weight of a shortest path in G from $v_{1,i}$ to $v_{d+1,j}$ after the k edges corresponding to the location increments of S.

We apply Lemma 3 to the distance sensitivity oracle of Theorem 2 for n times, where each time the graph is G and a distinct vertex of V_1 is chosen for the source s. We thus obtain n single source k edge-increase distance oracles. The time to construct all oracles is therefore $O((mn + n^2 \log n)(d \log n)^k)$. Now suppose that S is a set of k location increments, or, equivalently, we can view S as a set of k pairs (e_x, δ_x) for $x = 1, \ldots, k$ where e_x is an edge of G and δ_x is its increased weight. Then, by Theorem 2 and Lemma 3, each $C(S)[i,j]$ can be obtained in $O(2^k k \log^k n)$ time by querying the oracle corresponding to the source $s = v_{1,i}$ with the query (S, j). We summarize our result in the following theorem.

Theorem 4. *Given $n \times n$ matrices $A_1, \ldots, A_d$ with entries in $\mathbb{R} \cup \infty$ and with a total of m non-infinity entries, one can construct in $O((mn + n^2 \log n)(d \log n)^k)$ time a data structure that given a query S of k location increments, computes (deterministically) the distance product $C(S)$ in $O(n^2 \cdot 2^k k \log^k n)$ time.*

References

[1] Noga Alon, Shiri Chechik, and Sarel Cohen. Deterministic combinatorial replacement paths and distance sensitivity oracles. In *46th ICALP*, pages 12:1–12:14, 2019.

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

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

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

[5] Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Compact and fast sensitivity oracles for single-source distances. In *24th ESA*, pages 13:1–13:14, 2016.

[6] Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *33rd STACS*, pages 18:1–18:14, 2016.

[7] S. Chechik, M. Langberg, D. Peleg, and L. Roditty. f-sensitivity distance oracles and routing schemes. In *Proceedings of the 18th annual European Symposium on Algorithms (ESA)*, pages 84–96, 2010.

[8] Shiri Chechik and Sarel Cohen. Near optimal algorithms for the single source replacement paths problem. In *13th SODA*, pages 2090–2109, 2019.

[9] Shiri Chechik, Sarel Cohen, Amos Fiat, and Haim Kaplan. $(1+\epsilon)$-approximate f-sensitive distance oracles. In *28th SODA*, pages 1479–1496, 2017.

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

[11] Camil Demetrescu and Mikkel Thorup. Oracles for distances avoiding a link-failure. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 838–843. Society for Industrial and Applied Mathematics, 2002.

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

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

[14] Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *53rd FOCS*, pages 748–757, 2012.

[15] 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)*, pages 343–354, 2003.

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

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

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

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

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

[21] L. Roditty and U. Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. In *32nd ICALP*, pages 249–260, 2005.

[22] Oren Weimann and Raphael Yuster. Replacement paths via fast matrix multiplication. In *51th FOCS*, pages 655–662, 2010.

[23] Oren Weimann and Raphael Yuster. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Trans. Algorithms*, 9(2):14:1–14:13, 2013.

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

[25] Virginia Vassilevska Williams. Faster replacement paths. In *22nd SODA*, pages 1337–1346, 2011.

[26] Raphael Yuster and Uri Zwick. Answering distance queries in directed graphs using fast matrix multiplication. In *46th FOCS*, pages 389–396, 2005.

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