Constructing a Distance Sensitivity Oracle in $O(n^{2.5794}M)$ Time

Yong Gu
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China

Hanlin Ren
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China

Abstract

We continue the study of *distance sensitivity oracles* (DSOs). Given a directed graph G with n vertices and edge weights in $\{1, 2, \ldots, M\}$, we want to build a data structure such that given any source vertex u, any target vertex v, and any failure f (which is either a vertex or an edge), it outputs the length of the shortest path from u to v not going through f. Our main result is a DSO with preprocessing time $O(n^{2.5794}M)$ and constant query time. Previously, the best preprocessing time of DSOs for directed graphs is $O(n^{2.7233}M)$, and even in the easier case of undirected graphs, the best preprocessing time is $O(n^{2.6865}M)$ [Ren, ESA 2020]. One drawback of our DSOs, though, is that it only supports distance queries but not path queries.

Our main technical ingredient is an algorithm that computes the inverse of a degree-d polynomial matrix (i.e. a matrix whose entries are degree-d univariate polynomials) modulo x^r. The algorithm is adapted from [Zhou, Labahn and Storjohann, *Journal of Complexity*, 2015], and we replace some of its intermediate steps with faster rectangular matrix multiplication algorithms.

We also show how to compute *unique* shortest paths in a directed graph with edge weights in $\{1, 2, \ldots, M\}$, in $O(n^{2.5286}M)$ time. This algorithm is crucial in the preprocessing algorithm of our DSO. Our solution improves the $O(n^{2.6865}M)$ time bound in [Ren, ESA 2020], and matches the current best time bound for computing all-pairs shortest paths.

2012 ACM Subject Classification Theory of computation → Shortest paths; Theory of computation → Design and analysis of algorithms

Keywords and phrases graph theory, shortest paths, distance sensitivity oracles

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.76

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2102.08569

Acknowledgements We would like to thank Ran Duan and Tianyi Zhang for helpful discussions during the initial stage of this research. We are grateful to anonymous reviewers for helpful comments. We would also like to thank an anonymous reviewer for suggesting the title of Section 5, and another anonymous reviewer for pointing out a subtle issue regarding the invertibility of polynomial matrices (and fixing the issue).

1 Introduction

In this paper, we consider the problem of constructing a *distance sensitivity oracle* (DSO). A DSO is a data structure that preprocesses a directed graph $G = (V, E)$ with n vertices and m edges, and supports queries of the following form: Given a source vertex u, a target vertex v, and a failure f (which can be either a vertex or an edge), output the length of the shortest path from u to v that does not go through f.

One motivation for constructing DSOs is the fact that real-life networks often suffer from failures. Consider a communication network among n servers. When a server u wants to send a message to another server v, the most efficient way would be to send the message along the

shortest path from u to v. However, if a failure happens in a server or a link between two servers, we would need to recompute the shortest path with the failure taken into account. It may be too slow to compute the shortest path from scratch each time a failure happens. A better solution is to construct a DSO for the communication network, and invoke the query algorithm of the DSO whenever a failure happens.

1.1 Related Work

The problem of constructing DSOs has received a lot of attention in the literature. A naïve solution is to precompute the answers for every possible query (u, v, f), but it requires $\Omega(n^2 m)$ space to store this DSO. Demetrescu et al. [11] constructed a DSO with $O(n^2 \log n)$ space that answers a query in constant time. However, the preprocessing time of the DSO in [11] is $O(mn^2 + n^3 \log n)$, which is inefficient for large networks. Subsequently, Bernstein and Karger improved the preprocessing time to $\tilde{O}(n^2\sqrt{m})$ [5], and finally $\tilde{O}(mn)$ [6].[1] The preprocessing time $\tilde{O}(mn)$ matches the current best time bound for the easier problem of computing *all-pairs shortest paths* (APSP), and it is conjectured that APSP requires $mn^{1-o(1)}$ time [20]. In this sense, the $\tilde{O}(mn)$ time bound of [6] is optimal. Duan and Zhang [14] improved the space complexity of the DSO to $O(n^2)$, eliminating the last $\log n$ factor, while preserving constant query time and $\tilde{O}(mn)$ preprocessing time.

However, for dense graphs (i.e. $m = \Theta(n^2)$) with edge weights in $[-M, M]$, it is possible to compute APSP in time faster than $\tilde{O}(mn) = \tilde{O}(n^3)$. The best APSP algorithm for undirected graphs runs in $\tilde{O}(n^\omega M)$ time [27, 30], and the best APSP algorithm for directed graphs runs in $O(n^{2.5286}M)$ time [4, 42]. (Here $\omega < 2.3728596$ is the exponent of matrix multiplication [2, 9, 19, 32, 37].) Therefore, it is natural to ask whether one can beat $\tilde{O}(n^3)$ preprocessing time for DSOs in this regime.

The answer turned out to be *yes*. Weimann and Yuster [36] showed that for any constant $0 < \alpha < 1$, there is a DSO with $\tilde{O}(n^{1-\alpha+\omega}M)$ preprocessing time and $\tilde{O}(n^{1+\alpha})$ query time. Subsequently, Grandoni and Williams [16] showed that for any constant $0 < \alpha < 1$, there is a DSO with $\tilde{O}(n^{\omega+1/2}M + n^{\omega+\alpha(4-\omega)}M)$ preprocessing time and $\tilde{O}(n^{1-\alpha})$ query time. Recently, Chechik and Cohen [8] constructed the first DSO that achieves both sub-cubic ($O(n^{2.873}M)$) preprocessing time and poly-logarithmic query time simultaneously. For the case that edge weights are positive, Ren [22] improved the previous results by presenting a much simpler DSO with $\tilde{O}(n^{2.7233}M)$ preprocessing time and constant query time.

Note that most DSOs mentioned above are randomized. Recently, there are also some efforts on derandomizing these DSOs, see e.g. [3, 23].

1.2 Our Results

Our main result is an improved DSO for directed graphs with integer edge weights in $[1, M]$. In particular, our DSO has preprocessing time $O(n^{2.5794}M)$ and constant query time.

▶ **Theorem 1.1** (Main). *Given as input a directed graph $G = (V, E)$ with edge weights in $\{1, 2, \ldots, M\}$, we can construct a DSO with $O(n^{2.5794}M)$ preprocessing time and constant query time. With high probability over the randomized preprocessing algorithm, the DSO answers every possible query correctly.*

[1] $\tilde{O}$ hides polylog(n) factors.

▶ Remark 1.2. Our preprocessing algorithm uses fast *rectangular* matrix multiplication algorithms. To express our time bound as a function of ω, we could also simulate rectangular matrix multiplications by square matrix multiplications, e.g. multiply an $n \times m$ matrix and an $m \times n$ matrix by $\lceil m/n \rceil$ square matrix multiplications of dimension n. In this case, the preprocessing time becomes $\tilde{O}(n^{2+1/(4-\omega)}M) < O(n^{2.6146}M)$.

▶ Remark 1.3 (Comparison with Prior Works). The biggest advantage of our DSO is, of course, its fast preprocessing algorithm. In fact, the preprocessing time bound is only an $O(n^{0.051})$ factor away from the current best time bound for APSP. Our DSO is also the first one to break a barrier of $\tilde{\Omega}(n^{8/3})$ preprocessing time, while keeping constant query time.[2] However, our DSO has two drawbacks. First, it can only return the length of the shortest path. It does not suggest an efficient way to produce this path. Second, it does not support negative edge weights.

We highlight two technical ingredients that are crucial for the preprocessing algorithm of our DSO.

Inverting a polynomial matrix modulo x^r

Let r be an integer parameter, and $\mathbf{F}$ be a polynomial matrix of degree d (i.e. each entry of $\mathbf{F}$ is a degree-d polynomial over some formal variable x) that is invertible. We show how to compute $\mathbf{F}^{-1} \bmod x^r$ in time

$$\tilde{O}(dn^\omega) + (r^2/d) \cdot \mathsf{MM}(n, nd/r, nd/r) \cdot n^{o(1)}.$$

(That is, we only preserve the monomials in $\mathbf{F}^{-1}$ with degrees at most $r-1$.) Here, $\mathsf{MM}(n_1, n_2, n_3)$ is the time complexity of multiplying an $n_1 \times n_2$ matrix and an $n_2 \times n_3$ matrix.

It is shown in [40] that we can compute the full $\mathbf{F}^{-1}$ (instead of $\mathbf{F}^{-1} \bmod x^r$) in $\tilde{O}(n^3 d)$ time. We examine their algorithm carefully, and adapt it to our case where we only want to compute $\mathbf{F}^{-1} \bmod x^r$. We modulo each polynomial in the intermediate steps of the algorithm by x^r, and use fast rectangular matrix multiplication to speed up the algorithm.

▶ **Theorem 1.4.** *Let r be an integer, $\mathbb{F}$ be a finite field. Let $\mathbf{F} \in \mathbb{F}[x]^{n \times n}$ be an $n \times n$ matrix over the ring of univariate polynomials $\mathbb{F}[x]$, and let $d \geq 1$ be an upper bound on the degrees of entries of $\mathbf{F}$. If $\mathbf{F}$ is invertible over $(\mathbb{F}[x]/\langle x^r \rangle)^{n \times n}$, the number of field operations to compute $\mathbf{F}^{-1} \bmod x^r$ is at most*

$$\tilde{O}(dn^\omega) + (r^2/d) \cdot \mathsf{MM}(n, nd/r, nd/r) \cdot n^{o(1)}.$$

▶ Remark 1.5. The idea of using polynomial matrices to capture distances is a common technique in graph algorithms. It has found many applications in static algorithms [24], fault-tolerant algorithms [35], and dynamic algorithms [25, 33, 34].

[2] There are three previous DSOs with both sub-cubic preprocessing time and constant query time: [16], [8], and [22]. (The query time of the first two DSOs can be brought down to constant using Observation 2.1 of [22]. In the case of [16], this increases the preprocessing time by an additive factor of $\tilde{O}(n^{3-\alpha})$.) Even when $\omega = 2$, the preprocessing time bounds of these DSOs are $\tilde{O}(n^{8/3})$ (setting α appropriately), $\tilde{O}(n^{14/5})$, and $\tilde{O}(n^{8/3})$ respectively.

Computing consistent shortest path trees

Our DSO needs to invoke [22, Observation 2.1] (see also [6]), which needs a *consistent* set of (incoming and outgoing) shortest path trees rooted at each vertex. Here, by *consistent*, we mean that for every pair of vertices u, v and any two shortest path trees T_1 and T_2 (from the $2n$ trees; recall they are *directed* rooted trees), if u can reach v in both T_1 and T_2, then the $u \rightsquigarrow v$ paths in T_1 and T_2 are the same path. In other words, we want to specify a *unique* shortest path between each pair of vertices, such that for every vertex v, the shortest paths starting from v (or ending at v, respectively) form a tree.

Note that this problem is quite nontrivial in small-weighted graphs. There may be many shortest paths between two vertices, and it is not obvious how to pick one shortest path for each vertex pair, while guaranteeing consistency. Also, we cannot randomly perturb the edge weights by small values, as that would break the property that edge weights are small integers. It is also unclear how to construct such a set of shortest path trees from the APSP algorithm in [42]. Previously, combining ideas in [10, Section 3.4] and an algorithm in [13], [22] showed how to compute such shortest path trees in $\tilde{O}(n^{(3+\omega)/2}M) \leq O(n^{2.6865}M)$ time; unfortunately, this time bound is worse than our claimed time bound $O(n^{2.5794}M)$ in Theorem 1.1.

In this paper, we show how to construct consistent shortest paths trees in $O(n^{2.5286}M)$ time, matching the currently best time bound for APSP [42]. Below is an informal statement, see Theorem 5.1 for the precise version.

▶ **Theorem 1.6** (Informal Version). *Given a directed graph $G = (V, E)$ with edge weights in $\{1, 2, \ldots, M\}$, we can compute a set of incoming and outgoing shortest path trees rooted at each vertex that are consistent, in $O(n^{2.5286}M)$ time.*

1.3 Warm-Up: DSO in $\tilde{O}(n^{(3+\omega)/2}M)$ Preprocessing Time

Actually, the ideas in [35] of maintaining the *adjoint* of the *symbolic adjacency matrix* (see Section 3), together with ideas in [22], already give us a DSO with $\tilde{O}(n^{(3+\omega)/2}M)$ preprocessing time and constant query time. As a warm-up, we briefly describe this DSO before we proceed into the details of Theorem 1.1.

An *r-truncated DSO* [22] is a DSO that only needs to be correct for the queries (u, v, f) whose answer (i.e. length of the corresponding shortest path) is at most r. If the answer is greater than r, it should return r instead. In what follows, we will describe how to construct an r-truncated DSO in $\tilde{O}(rn^{\omega})$ preprocessing time and $\tilde{O}(r)$ query time. Using techniques in [22] (see also Section 3.3), this implies a DSO with $\tilde{O}(n^{(3+\omega)/2}M)$ preprocessing time and constant query time.

Let $\mathbb{F}$ be a sufficiently large finite field, and $\mathbf{A}$ be the following matrix. For every vertices u, v, if there is an edge from u to v with weight l, then let $\mathbf{A}_{u,v} = a_{u,v}x^l$, where $a_{u,v}$ is a random element in $\mathbb{F}$, and x is an indeterminate. Furthermore, for every vertex v, let $\mathbf{A}_{v,v} = 1$. It is well-known [24] that with high probability over the choices of $a_{u,v}$, the *adjoint* matrix of $\mathbf{A}$ encodes the shortest path information of the input graph, as follows. Let $\mathrm{adj}(\mathbf{A})$ be the adjoint matrix of $\mathbf{A}$, and u, v be two vertices, then the lowest degree of $\mathrm{adj}(\mathbf{A})_{u,v}$ is exactly the distance from u to v. For example, if $\mathrm{adj}(\mathbf{A})_{u,v} = 7x^8 + 6x^5 - 9x^4$, then the distance from u to v is 4.

A big advantage of the adjoint matrix, exploited in [35] and also this work, is that it is easy to perform *low-rank* updates, by the Sherman-Morrison-Woodbury formula (see Theorem 3.2). Given a matrix $\mathbf{A}$, its adjoint $\mathrm{adj}(\mathbf{A})$, and a low-rank matrix $\mathbf{B}$, we can compute a specific element of $\mathrm{adj}(\mathbf{A}+\mathbf{B})_{u,v}$, in time much faster than brute force. Therefore,

we answer a query (u, v, f) as follows: We first express the failure as a *rank-one* matrix $\mathbf{F}$, such that $\mathbf{A} + \mathbf{F}$ is the matrix corresponding to the graph with f removed. Then we can compute $\text{adj}(\mathbf{A} + \mathbf{F})_{u,v}$ quickly. Given this element (a polynomial over $\mathbb{F}$), we can easily compute the answer to the query.

What is the time complexity of this DSO? Recall that we only want to construct an r-truncated DSO, so we can modulo every entry in the process of computing $\text{adj}(\mathbf{A})$ by the polynomial x^r. Every arithmetic operation in the commutative ring $\mathbb{F}[x]/x^r$ only takes $\tilde{O}(r)$ time. Computing the adjoint of a matrix reduces to inverting that matrix, which takes $\tilde{O}(n^\omega)$ arithmetic operations [7]. Therefore it takes $\tilde{O}(rn^\omega)$ time to compute $\text{adj}(\mathbf{A}) \bmod x^r$. A close inspection of the Sherman-Morrison-Woodbury formula shows that each query can be completed in $O(1)$ arithmetic operations, i.e. $\tilde{O}(r)$ time.

The $\tilde{O}(rn^\omega)$-time algorithm for inverting a polynomial matrix modulo x^r is not optimal; the time bound in Theorem 1.4 is better. In Section 4, we use fast rectangular matrix multiplication algorithms to speed up the algorithm in [40], obtaining a faster algorithm for inverting polynomial matrices modulo x^r.

2 Preliminaries

In this paper, we say an event happens *with high probability* (w.h.p.) if it happens with probability at least $1 - 1/n^c$, for a constant c that can be made arbitrarily large. Our DSOs (or r-truncated DSOs) will have a randomized preprocessing algorithm and a deterministic query algorithm. We say a DSO is *correct with high probability* if w.h.p. over its (randomized) preprocessing algorithm, it answers every possible query (u, v, f) correctly.

Notation

We use the following notation in [12, 22].

- Let p be a path, we use $|p|$ to denote the number of edges in p, and use $\|p\|$ to denote the length of p (i.e. total weight of edges in p).
- Let u, v be two vertices, we define $\|uv\|$ as the length of the shortest path from u to v. Furthermore, let f be a failure (which is either an edge or a vertex), we define $\|uv \diamond f\|$ as the length of the shortest path from u to v that does not go through f.
- Let u, v be two vertices, we define $|uv|$ as the number of edges in the shortest path from u to v. In the case that there are many shortest paths from u to v, it turns out that the following definition will be convenient in Section 5: We define $|uv|$ as the *largest* number of edges in any shortest path from u to v.

Fast matrix multiplication

Let ω be the exponent of matrix multiplication; the current best upper bound is $\omega \leq 2.3728596$ [2]. For positive integers n_1, n_2, n_3, let $\mathsf{MM}(n_1, n_2, n_3)$ denote the minimum number of arithmetic operations needed to multiply an $n_1 \times n_2$ matrix and an $n_2 \times n_3$ matrix. We define $\omega(a, b, c)$ to be the exponent of multiplying an $n^a \times n^b$ matrix and an $n^b \times n^c$ matrix, i.e.

$$\omega(a, b, c) = \inf\{w : \mathsf{MM}(n^a, n^b, n^c) = O(n^w)\}.$$

It is a classical result that $\omega(1, 1, \lambda) = \omega(1, \lambda, 1) = \omega(\lambda, 1, 1)$ for any real number $\lambda > 0$ [21]; we denote $\omega(\lambda) = \omega(1, 1, \lambda)$.

We will need the following lemmas about the exponent of rectangular matrix multiplication. We refer the reader to the full version of this paper for their proofs.

▶ **Lemma 2.1.** *Let a, b, c, r be positive real numbers, then $r + \omega(a, b, c) \leq \omega(a, b + r, c + r)$.*

▶ **Lemma 2.2.** *Consider the function $f(\tau) = \omega(1, 1 - \tau, 1 - \tau)$, where $\tau \in [0, 1]$. Then $\tau + f(\tau)$ is monotonically non-increasing in τ, and $2\tau + f(\tau)$ is monotonically non-decreasing in τ.*

Polynomial operations

Let $p, q \in \mathbb{F}[x]$ be two polynomials of degree d. It is easy to compute $p + q$ or $p - q$ in $O(d)$ field operations. We can also compute $p \cdot q$ in $\tilde{O}(d)$ field operations using fast Fourier transform. (Here, $\tilde{O}$ hides polylog(d) factors.) When p is invertible, it is also possible to compute $p^{-1} \bmod x^d$ in $\tilde{O}(d)$ field operations [1, Section 8.3].

3 Constructing a DSO in $O(n^{2.5794}M)$ Time

In this section, we show how to preprocess a distance sensitivity oracle in $O(n^{2.5794}M)$ time, such that every query can be answered in constant time. Our preprocessing algorithm is randomized; with high probability over the preprocessing algorithm, the query algorithm always returns the correct answer.

3.1 Preliminaries

First, our preprocessing algorithm will use the following algorithm for inverting a polynomial matrix. A sketch of this algorithm will be given in Section 4.

▶ **Theorem 1.4.** *Let r be an integer, $\mathbb{F}$ be a finite field. Let $\mathbf{F} \in \mathbb{F}[x]^{n \times n}$ be an $n \times n$ matrix over the ring of univariate polynomials $\mathbb{F}[x]$, and let $d \geq 1$ be an upper bound on the degrees of entries of $\mathbf{F}$. If $\mathbf{F}$ is invertible over $(\mathbb{F}[x]/\langle x^r \rangle)^{n \times n}$, the number of field operations to compute $\mathbf{F}^{-1} \bmod x^r$ is at most*

$$\tilde{O}(dn^\omega) + (r^2/d) \cdot \mathsf{MM}(n, nd/r, nd/r) \cdot n^{o(1)}.$$

Let G be a directed graph whose edge weights are integers in $[1, M]$. We define its *symbolic adjacency matrix* $\mathsf{SA}(G)$ as (see [24])

$$\mathsf{SA}(G)_{i,j} = \begin{cases} 1 & \text{if } i = j, \\ z_{i,j}x^l & \text{if there is an edge from } i \text{ to } j \text{ with weight } l \text{ in } G, \\ 0 & \text{otherwise,} \end{cases}$$

where $z_{i,j}$ are unique variables corresponding to edges of G.

It will be inefficient to deal with these variables $z_{i,j}$, therefore we will pick a suitably large field $\mathbb{F}$, and substitute each variable $z_{i,j}$ by a random element in $\mathbb{F}$. However, we still keep the indeterminate x. Now, let $\mathbf{Z}$ be a matrix where each $\mathbf{Z}_{i,j} \in \mathbb{F}$, we will use $\mathsf{SA}_{\mathbf{Z}}(G)$ to denote the matrix $\mathsf{SA}(G)$ with each formal variable $z_{i,j}$ substituted by the field element $\mathbf{Z}_{i,j}$. Note that $\mathsf{SA}_{\mathbf{Z}}(G)$ is a polynomial matrix where every entry is a polynomial over x with degree at most M.

We recall the definition of *adjoint* matrix that will be crucial to our algorithm. Let $\mathbf{A}$ be an $n \times n$ matrix, $i, j \in [n]$. We denote by $\mathbf{A}^{i,j}$ the matrix $\mathbf{A}$ with every element in the i-th row and the j-th column set to zero, except that $(\mathbf{A}^{i,j})_{i,j} = 1$. The adjoint matrix of $\mathbf{A}$, denoted as $\mathrm{adj}(\mathbf{A})$, is an $n \times n$ matrix such that $\mathrm{adj}(\mathbf{A})_{i,j} = \det(\mathbf{A}^{j,i})$ for every $i, j \in [n]$. A basic fact about $\mathrm{adj}(\mathbf{A})$ is that if $\det(\mathbf{A}) \neq 0$, then $\mathrm{adj}(\mathbf{A}) = \det(\mathbf{A}) \cdot \mathbf{A}^{-1}$.

There is a close relationship between the distances in the graph G, and the entries in the adjoint of $\mathsf{SA}(G)$. Let p be a multivariate polynomial, we define $\deg_x^*(p)$ as the lowest degree of the variable x in any monomial of p. If $p = 0$, then we define $\deg_x^*(p) := +\infty$. We have:

▶ **Theorem 3.1** ([24, Lemma 4]). *Let G be a directed graph with positive integer weights, i, j be two vertices. Then the distance from i to j in G is $\deg_x^*(\mathrm{adj}(\mathsf{SA}(G))_{i,j})$.*

We need the following theorem that allows us to maintain the adjoint of a matrix under *rank*-1 queries. (This theorem is a special case of [35, Lemma 1.6].)

▶ **Theorem 3.2.** *Let $\mathcal{R}$ be an arbitrary commutative ring, $\mathbf{A} \in \mathcal{R}^{n\times n}$ be an invertible matrix, $\mathbf{u}, \mathbf{v} \in \mathcal{R}^n$ be column vectors, and $\gamma = 1 + \mathbf{v}^\mathsf{T}\mathbf{A}^{-1}\mathbf{u}$. Suppose γ is invertible, then $\mathbf{A} + \mathbf{u}\mathbf{v}^\mathsf{T}$ is also invertible, and*

$$\mathrm{adj}(\mathbf{A} + \mathbf{u}\mathbf{v}^\mathsf{T}) = \det(\mathbf{A})(\gamma\mathbf{A}^{-1} - (\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^\mathsf{T}\mathbf{A}^{-1})).$$

Proof Sketch. By the matrix determinant lemma, we have

$$\det(\mathbf{A} + \mathbf{u}\mathbf{v}^\mathsf{T}) = \gamma \cdot \det(\mathbf{A}).$$

Since γ is invertible, we can use the Sherman-Morrison-Woodbury formula [29, 38]:

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^\mathsf{T})^{-1} = \mathbf{A}^{-1} - \gamma^{-1}(\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^\mathsf{T}\mathbf{A}^{-1}).$$

The theorem is proved by multiplying the above two formulas together. ◀

We need the Schwartz-Zippel lemma that guarantees the correctness of our randomized algorithm.

▶ **Theorem 3.3** (Schwartz-Zippel Lemma, [26, 41]). *Let $p(x_1, x_2, \ldots, x_m)$ be a non-zero polynomial of (total) degree d over a field $\mathbb{F}$. Let S be a finite subset of $\mathbb{F}$, and $r_1, r_2, \ldots, r_m$ be independently and uniformly sampled from S. Then*

$$\Pr[p(r_1, r_2, \ldots, r_m) = 0] \le \frac{d}{|S|}.$$

We also need the following algorithm that computes the determinant of a polynomial matrix.

▶ **Theorem 3.4** ([18, 31]). *Let $\mathbf{B} \in \mathbb{F}[x]^{n\times n}$ be a matrix of degree at most d, then we can compute $\det(\mathbf{B})$ in $\tilde{O}(dn^\omega)$ field operations.*

3.2 Constructing an r-Truncated DSO

Recall that for a failure f (which is either a vertex or an edge), $\|uv \diamond f\|$ denotes the length of the shortest path from u to v that avoids f. An *r-truncated* DSO, as defined in [22], is a DSO that given a query (u, v, f), outputs the value $\min\{\|uv \diamond f\|, r\}$. The main result of this subsection is that given an integer r and an input graph G, an r-truncated DSO can be constructed in time

$$\tilde{O}(n^\omega M) + r^2/M \cdot \mathsf{MM}(n, nM/r, nM/r) \cdot n^{o(1)}.$$

Preprocessing algorithm

Let C be a large enough constant. First, we choose a prime $p \in [n^C, 2n^C]$ and let $\mathbb{F} = \mathbb{Z}_p$. Then we let $\mathbf{Z}$ be an $n \times n$ matrix over $\mathbb{F}$, where every $\mathbf{Z}_{i,j}$ is sampled independently from $\mathbb{F}$ uniformly at random. We substitute $\mathbf{Z}$ into $\mathsf{SA}(G)$ to obtain the matrix $\mathsf{SA}_{\mathbf{Z}}(G)$. Recall that each element of $\mathsf{SA}_{\mathbf{Z}}(G)$ is a polynomial over x with coefficients in $\mathbb{F}$, whose degree is at most M. Then we compute $\mathsf{SA}_{\mathbf{Z}}(G)^{-1}$ and $\det(\mathsf{SA}_{\mathbf{Z}}(G))$ using Theorem 1.4 and Theorem 3.4 respectively.

Since we only want an r-truncated DSO, we only need to compute $\mathsf{SA}_{\mathbf{Z}}(G)^{-1}$ modulo x^r, i.e. we only preserve the monomials with degree less than r in every entry of $\mathsf{SA}_{\mathbf{Z}}(G)^{-1}$. Note that $\mathsf{SA}_{\mathbf{Z}}(G)$ is of the form $\mathbf{I} + x\mathbf{M}$ for some matrix $\mathbf{M} \in \mathbb{F}[x]^{n\times n}$, therefore its determinant is of the form $1 + x \cdot p(x)$ for some polynomial $p(x)$. As the determinant is invertible modulo x^r, $\mathsf{SA}_{\mathbf{Z}}(G)$ is also invertible modulo x^r. By Theorem 1.4, we can compute $\mathsf{SA}_{\mathbf{Z}}(G)^{-1} \bmod x^r$ in time

$$\tilde{O}(n^\omega M) + (r^2/M) \cdot \mathsf{MM}(n, nM/r, nM/r) \cdot n^{o(1)}.$$

By Theorem 3.4, we can compute $\det(\mathsf{SA}_{\mathbf{Z}}(G))$ in $\tilde{O}(n^\omega M)$ time. Again, we only need to store the polynomial $\det(\mathsf{SA}_{\mathbf{Z}}(G)) \bmod x^r$. This concludes the preprocessing algorithm.

For the following query algorithms, we use $\mathbf{e}_i$ to denote the i-th standard unit vector, i.e. $(\mathbf{e}_i)_i = 1$, and $(\mathbf{e}_i)_j = 0$ for every index $j \neq i$.

Query algorithm for an edge failure

A query consists of vertices $u, v \in V$ and a failed edge e. We assume that e goes from vertex a to vertex b, and has weight l. Let G' be the graph obtained by removing e from G, then we have $\mathsf{SA}(G') = \mathsf{SA}(G) + \mathbf{u}\mathbf{v}^\mathsf{T}$, where $\mathbf{u} = \mathbf{e}_a$ and $\mathbf{v} = -z_{a,b}x^l\mathbf{e}_b$. Let

- $\gamma = 1 + \mathbf{v}^\mathsf{T}\mathsf{SA}(G)^{-1}\mathbf{u} = 1 - z_{a,b}x^l\mathsf{SA}(G)^{-1}_{b,a}$,
- $\beta = (\mathsf{SA}(G)^{-1}\mathbf{u}\mathbf{v}^\mathsf{T}\mathsf{SA}(G)^{-1})_{u,v} = -\mathsf{SA}(G)^{-1}_{u,a}z_{a,b}\mathsf{SA}(G)^{-1}_{b,v}x^l$, and
- $\alpha = \det(\mathsf{SA}(G))(\gamma \cdot \mathsf{SA}(G)^{-1}_{u,v} - \beta)$,

then by Theorem 3.2, we have $\alpha = \mathrm{adj}(\mathsf{SA}(G'))_{u,v}$. (Note that since $l \geq 1$, γ is always invertible.)

Query algorithm for a vertex failure

A query consists of vertices $u, v \in V$ and a failed vertex $f \in V$. It suffices to remove every outgoing edge from f (and we do not need to also remove incoming edges to f), as f already cannot appear as an intermediate vertex in every path from u to v. Therefore, we need to compute $\mathrm{adj}(\mathsf{SA}(G'))_{u,v}$, where G' is obtained by removing all outgoing edges from f in G. Let $\mathbf{u} = \mathbf{e}_f$, and $\mathbf{v}$ be the negation of the transpose of the f-th row of $\mathsf{SA}(G)$, except that $\mathbf{v}_f = 0$, i.e.,

$$\mathbf{v}_j = \begin{cases} -z_{f,j}x^l & \text{if there is an edge from } f \text{ to } j \text{ with weight } l \text{ in } G, \\ 0 & \text{otherwise,} \end{cases}$$

It is easy to see $\mathsf{SA}(G') = \mathsf{SA}(G) + \mathbf{u}\mathbf{v}^\mathsf{T}$. To compute $\mathrm{adj}(\mathsf{SA}(G'))_{u,v}$ using Theorem 3.2, we let

- $\gamma = 1 + \mathbf{v}^\mathsf{T}\mathsf{SA}(G)^{-1}\mathbf{u}$. Note that $(\mathbf{e}_f - \mathbf{v})^\mathsf{T}$ is exactly the f-th row of $\mathsf{SA}(G)$, so $(\mathbf{e}_f - \mathbf{v})^\mathsf{T}\mathsf{SA}(G)^{-1} = \mathbf{e}_f^\mathsf{T}$, and $\mathbf{v}^\mathsf{T}\mathsf{SA}(G)^{-1} = \mathbf{e}_f^\mathsf{T}\mathsf{SA}(G)^{-1} - \mathbf{e}_f^\mathsf{T}$. We have $\gamma = 1 + \mathbf{e}_f^\mathsf{T}\mathsf{SA}(G)^{-1}\mathbf{u} - \mathbf{e}_f^\mathsf{T}\mathbf{u} = \mathsf{SA}(G)^{-1}_{f,f}$;

- $\beta = (\mathsf{SA}(G)^{-1}\mathbf{u}\mathbf{v}^{\mathsf{T}}\mathsf{SA}(G)^{-1})_{u,v} = (\mathbf{e}_u^{\mathsf{T}}\mathsf{SA}(G)^{-1}\mathbf{u})(\mathbf{v}^{\mathsf{T}}\mathsf{SA}(G)^{-1}\mathbf{e}_v)$
  $= \mathsf{SA}(G)^{-1}_{u,f}(\mathbf{e}_f^{\mathsf{T}}\mathsf{SA}(G)^{-1}\mathbf{e}_v) = \mathsf{SA}(G)^{-1}_{u,f}\mathsf{SA}(G)^{-1}_{f,v}$;
- and $\alpha = \det(\mathsf{SA}(G))(\gamma \cdot \mathsf{SA}(G)^{-1}_{u,v} - \beta)$,

then we have $\alpha = \mathrm{adj}(\mathsf{SA}(G'))_{u,v}$. (Note that γ is always invertible since the constant term of $\mathsf{SA}(G)^{-1}_{f,f}$ must be 1.)

In the actual query algorithm, we will substitute each formal variable $z_{i,j}$ by $\mathbf{Z}_{i,j}$. Let $\gamma_{\mathbf{Z}}$ denote the resulting polynomial after this substitution. Note that $\gamma_{\mathbf{Z}}$ is a polynomial in $\mathbb{F}[x]$. Similarly we can define $\beta_{\mathbf{Z}}$ and $\alpha_{\mathbf{Z}}$. If $\alpha_{\mathbf{Z}} \not\equiv 0 \pmod{x^r}$, then our query algorithm outputs $\deg^*_x(\alpha_{\mathbf{Z}})$; otherwise it outputs r.

From the above formulas, we can compute $\gamma_{\mathbf{Z}}$, $\beta_{\mathbf{Z}}$, and $\alpha_{\mathbf{Z}}$ in $O(1)$ arithmetic operations over polynomials. Note that we only need to compute these polynomials modulo x^r, so each such arithmetic operation takes $\tilde{O}(r)$ time. The total query time is thus $\tilde{O}(r)$.

▶ Remark 3.5 (Query Algorithm for Undirected Graphs). Our r-truncated DSO can also deal with undirected graphs, but the details are a bit different from the case of directed graphs. To remove an undirected edge, we need to update two entries in $\mathsf{SA}(G)$, which corresponds to a rank-2 update to $\mathsf{SA}(G)$. To remove a vertex, we need to update one row and one column in $\mathsf{SA}(G)$, which is also a rank-2 update to $\mathsf{SA}(G)$. Therefore, we need to use the rank-2 version of Theorem 3.2 (see [35, Lemma 1.6]). Actually, our r-truncated DSOs also support deleting f failures, and the query time is $\tilde{O}(f^\omega r)$. We omit the details here and refer the interested readers to [35].

▶ **Theorem 3.6.** *For every integer r, we can construct an r-truncated DSO with preprocessing time*

$$\tilde{O}(n^\omega M) + r^2/M \cdot \mathsf{MM}(n, nM/r, nM/r) \cdot n^{o(1)},$$

and query time $\tilde{O}(r)$. Our r-truncated DSO is correct w.h.p.

(Recall that by saying our r-truncated DSO is correct w.h.p, we mean that w.h.p. over its randomized preprocessing algorithm, it answers every query correctly.)

Proof of Theorem 3.6. We only need to prove the correctness of our r-truncated DSO. Consider a query (u, v, f) where f is an edge or a vertex, and let G' be the graph obtained by removing f from G. By Theorem 3.2, we have $\alpha_{\mathbf{Z}} = \mathrm{adj}(\mathsf{SA}_{\mathbf{Z}}(G'))_{u,v}$. (Note that the constant term of $\gamma_{\mathbf{Z}}$ is always 1, so $\gamma_{\mathbf{Z}}$ is always invertible.)

If $\|uv \diamond f\| \geq r$, then by Theorem 3.1, $\mathrm{adj}(\mathsf{SA}(G'))_{u,v}$ must be a polynomial whose minimum degree over x is at least r. In this case, we have $\alpha_{\mathbf{Z}} \equiv 0 \pmod{x^r}$ for every $\mathbf{Z}$. Therefore, our algorithm returns r, which is correct.

If $\|uv \diamond f\| = k < r$, then by Theorem 3.1, $\mathrm{adj}(\mathsf{SA}(G'))_{u,v}$ must be a polynomial whose minimum degree is exactly k. In this case, the coefficient of x^k in α is a polynomial of $z_{i,j}$ with (total) degree at most n. (This is because $\mathrm{adj}(\mathsf{SA}(G'))_{u,v}$ is the determinant of a certain $n \times n$ matrix in which every entry has total degree at most one in the variables $z_{i,j}$.) If this polynomial is nonzero at $\mathbf{Z}$, then $\deg^*_x(\alpha_{\mathbf{Z}}) = k$ and our query algorithm is correct. By Theorem 3.3, this polynomial is 0 with probability at most $1/n^{C-1}$. Therefore, our query algorithm returns the correct answer k with probability at least $1 - 1/n^{C-1}$.

In conclusion, for every fixed query (u, v, f), our query algorithm is correct with probability $1-1/n^{C-1}$ over the choice of $\mathbf{Z}$. By a union bound over $O(n^4)$ possible queries, the probability (over our randomized preprocessing algorithm) that every query is answered correctly is at least $1 - 1/\Theta(n^{C-5})$, which is a high probability. ◀

3.3 Constructing the Full DSO

Now we have constructed an r-truncated DSO, which we denote by $\mathcal{D}^{\mathsf{start}}$. In this subsection, we will extend it to a *full* DSO using the techniques in [22]. Specifically, we use the following two algorithms from [22].

The first algorithm transforms an (r-truncated) DSO with a possibly large query time into an (r-truncated) DSO with query time $O(1)$. More precisely:

▶ **Lemma 3.7** ([22, Observation 2.1]). *Given an r-truncated DSO $\mathcal{D}$ with preprocessing time P and query time Q, we can build an r-truncated DSO $\mathsf{Fast}(\mathcal{D})$ with query time $O(1)$ which is correct w.h.p. The preprocessing algorithm of $\mathsf{Fast}(\mathcal{D})$ is as follows:*

- *It needs the all-pairs distance matrix of the input graph G, as well as the set of* consistent *(incoming and outgoing) shortest path trees rooted at each vertex in G. By Theorem 1.6, these shortest path trees can be computed in $O(n^{2.5286}M)$ time. For details, see Section 5.*
- *It invokes the preprocessing algorithm of $\mathcal{D}$ on the input graph G once, and makes $\tilde{O}(n^2)$ queries to $\mathcal{D}$. The preprocessing time is $P + \tilde{O}(n^2)Q$.*

The second algorithm we use is implicit in the argument of [22, Section 2.3]. We formalize it as the following lemma.

▶ **Lemma 3.8.** *Given an r-truncated DSO $\mathcal{D}$ with preprocessing time P and query time $O(1)$, we can build a $(3/2)r$-truncated DSO $\mathsf{Extend}(\mathcal{D})$ with preprocessing time $P + O(n^2)$ and query time $\tilde{O}(nM/r)$. The new DSO is correct w.h.p.*

Now, we are ready to explain our algorithm to build a full DSO. Given an r-truncated DSO $\mathcal{D}^{\mathsf{start}}$, we first obtain an r-truncated DSO $\mathcal{D}_0$ with query time $O(1)$ by applying Lemma 3.7.

Let $i^\star = \lfloor \log_{3/2}(nM/r) \rfloor$. For every $0 \le i \le i^\star$, we construct an $r(3/2)^{i+1}$-truncated DSO $\mathcal{D}_{i+1}$ by applying Lemma 3.8 and Lemma 3.7 sequentially on $\mathcal{D}_i$, i.e. $\mathcal{D}_{i+1} = \mathsf{Fast}(\mathsf{Extend}(\mathcal{D}_i))$. Let the resulting DSO be $\mathcal{D}^{\mathsf{final}} = \mathcal{D}_{i^\star+1}$, since $r(3/2)^{i^\star+1} \ge nM$, $\mathcal{D}^{\mathsf{final}}$ is a full DSO.

We can also summarize our construction algorithm in one formula:

$$\mathcal{D}^{\mathsf{final}} = \underbrace{\mathsf{Fast}(\mathsf{Extend}(\mathsf{Fast}(\mathsf{Extend}(\cdots \mathsf{Fast}(\mathcal{D}^{\mathsf{start}})))))}_{O(\log(nM/r)) \text{ times}}.$$

Complexity of our DSO

Let $r = Mn^\alpha$, where $\alpha \in [0, 1]$ is a parameter to be determined. By Theorem 3.6, the preprocessing time of $\mathcal{D}^{\mathsf{start}}$ is

$$\tilde{O}(n^\omega M) + r^2/M \cdot \mathsf{MM}(n, nM/r, nM/r) \cdot n^{o(1)} \le \tilde{O}(n^\omega M) + n^{2\alpha+\omega(1,1-\alpha,1-\alpha)+o(1)}M,$$

and the query time of $\mathcal{D}^{\mathsf{start}}$ is $\tilde{O}(r) = \tilde{O}(n^\alpha M)$. By Lemma 3.7, the preprocessing time of $\mathcal{D}_0$ is

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

Now consider the preprocessing algorithm of $\mathcal{D}^{\mathsf{final}}$. We need to compute the all-pairs distance matrix and in/out shortest path trees of G as required by Lemma 3.7, which takes $\tilde{O}(n^{2+\mu}M)$ time by Theorem 1.6. We also need to run the preprocessing algorithm of $\mathcal{D}_0$. Also, for every $0 \le i \le i^\star$, we need to preprocess the oracle $\mathcal{D}_{i+1}$, which takes $n^2 \cdot \tilde{O}(nM/(r(3/2)^{i+1})) = \tilde{O}\left(\frac{n^{3-\alpha}M}{(3/2)^i}\right)$ time.

Therefore, the preprocessing time of $\mathcal{D}^{\mathsf{final}}$ is:

$$\tilde{O}(n^{2+\alpha}M + n^{\omega}M + n^{2+\mu}M) + n^{2\alpha+\omega(1,1-\alpha,1-\alpha)+o(1)}M + \sum_{i=0}^{\lfloor \log_{3/2}(nM/r)\rfloor} \tilde{O}\left(\frac{n^{3-\alpha}M}{(3/2)^i}\right)$$
$$\leq n^{\max\{2+\alpha,2+\mu,3-\alpha,2\alpha+\omega(1,1-\alpha,1-\alpha)\}+o(1)}M.$$

Let $\alpha = 0.420645$, $\beta = \frac{1}{1-\alpha}$, then $1.5 < \beta < 1.75$. Recall that for any real number λ, $\omega(\lambda)$ is a shorthand for $\omega(1,1,\lambda)$. We have

$$\begin{aligned}
\omega(1,1-\alpha,1-\alpha) &= (1-\alpha)\omega(\beta) \\
&\leq (1-\alpha)\cdot\frac{(1.75-\beta)\omega(1.5)+(\beta-1.5)\omega(1.75)}{1.75-1.5} && (1)\\
&\leq 0.579355\cdot 4\cdot(0.023943\cdot\omega(1.5)+0.226058\cdot\omega(1.75)) \\
&\leq 1.738094. && (2)
\end{aligned}$$

Here, Equation (1) uses the convexity of the $\omega(\cdot)$ function [21], and Equation (2) uses the recent bounds in [15] that $\omega(1.5) \leq 2.796537$ and $\omega(1.75) \leq 3.021591$. We can see that

$$\max\{2+\alpha, 2+\mu, 3-\alpha, 2\alpha+\omega(1,1-\alpha,1-\alpha)\} = 2\alpha+\omega(1,1-\alpha,1-\alpha) \leq 2.579384.$$

By Lemma 3.7, the query time of $\mathcal{D}^{\mathsf{final}}$ is $O(1)$. Therefore, we can construct a DSO with $O(n^{2.5794}M)$ preprocessing time and $O(1)$ query time.

As the DSOs constructed in Lemma 3.7 always have size $\tilde{O}(n^2)$, our final DSO only occupies $\tilde{O}(n^2)$ space. However, we remark that the preprocessing algorithm of our DSO requires $\tilde{O}(rn^2) = O(n^{2.4207})$ space (in particular, to store $\mathsf{SA}_{\mathbf{Z}}(G)^{-1} \bmod x^r$).

4 Inverting a Polynomial Matrix Modulo x^r

As we see in Section 3, the algorithm in Theorem 1.4 for inverting a polynomial matrix modulo x^r is very crucial for our results.

▶ **Theorem 1.4.** *Let r be an integer, $\mathbb{F}$ be a finite field. Let $\mathbf{F} \in \mathbb{F}[x]^{n\times n}$ be an $n\times n$ matrix over the ring of univariate polynomials $\mathbb{F}[x]$, and let $d \geq 1$ be an upper bound on the degrees of entries of $\mathbf{F}$. If $\mathbf{F}$ is invertible over $(\mathbb{F}[x]/\langle x^r\rangle)^{n\times n}$, the number of field operations to compute $\mathbf{F}^{-1} \bmod x^r$ is at most*

$$\tilde{O}(dn^{\omega}) + (r^2/d)\cdot \mathsf{MM}(n, nd/r, nd/r)\cdot n^{o(1)}.$$

Our algorithm is essentially the algorithm in [40]. In fact, the only difference is that we only consider polynomials modulo x^r. This allows us to invert the polynomial matrix in time faster than $\tilde{O}(n^3 d)$ using fast rectangular matrix multiplication algorithms.

In this section, we provide a very brief exposition of the algorithm in [40], and justify the time bound in Theorem 1.4. A detailed description of the algorithm can be found in the full version.

Let $\mathbf{F}$ be an input polynomial matrix where each entry has degree at most d. Suppose $\mathbf{F}$ is invertible over $(\mathbf{F}[x]/\langle x^r\rangle)^{n\times n}$. We will compute a *kernel basis decomposition* of $\mathbf{F}$, which is a chain of matrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_{\log n}$ and a diagonal matrix $\mathbf{B}$, such that

$$\mathbf{F}^{-1} = \mathbf{A}_1\mathbf{A}_2\ldots\mathbf{A}_{\log n}\mathbf{B}^{-1}. \quad (3)$$

Then, to compute $\mathbf{F}^{-1}$, we simply multiply the above matrices. Note that $\mathbf{B}$ is a diagonal matrix, so its inverse is easy to compute.[3]

To start, we write $\mathbf{F} = \begin{bmatrix}\mathbf{F}_{\mathsf{U}}\\ \mathbf{F}_{\mathsf{D}}\end{bmatrix}$, where each $\mathbf{F}_{\mathsf{U}}$ or $\mathbf{F}_{\mathsf{D}}$ is an $(n/2) \times n$ matrix. Then we compute two $n \times (n/2)$ matrices $\mathbf{N}_{\mathsf{R}}$ and $\mathbf{N}_{\mathsf{L}}$ with full rank, such that $\mathbf{F}_{\mathsf{U}}\mathbf{N}_{\mathsf{R}} = \mathbf{0}$, and $\mathbf{F}_{\mathsf{D}}\mathbf{N}_{\mathsf{L}} = \mathbf{0}$. (This can be done by [39, Theorem 4.2].) Let $\mathbf{A}_1 = \begin{bmatrix}\mathbf{N}_{\mathsf{L}} & \mathbf{N}_{\mathsf{R}}\end{bmatrix}$, then $\mathbf{A}_1$ has full rank, and

$$\mathbf{F} \cdot \mathbf{A}_1 = \begin{bmatrix}\mathbf{F}_{\mathsf{U}}\mathbf{N}_{\mathsf{L}} & \mathbf{F}_{\mathsf{U}}\mathbf{N}_{\mathsf{R}}\\ \mathbf{F}_{\mathsf{D}}\mathbf{N}_{\mathsf{L}} & \mathbf{F}_{\mathsf{D}}\mathbf{N}_{\mathsf{R}}\end{bmatrix} = \begin{bmatrix}\mathbf{F}_{\mathsf{U}}\mathbf{N}_{\mathsf{L}} & \\ & \mathbf{F}_{\mathsf{D}}\mathbf{N}_{\mathsf{R}}\end{bmatrix}.$$

Therefore, $\mathbf{F} \cdot \mathbf{A}_1$ is a block diagonal matrix with two blocks, each of size $(n/2) \times (n/2)$. We can then recursively invoke the kernel basis decomposition of these two blocks, and form the matrices $\mathbf{A}_2, \ldots, \mathbf{A}_{\log n}$. The diagonal matrix $\mathbf{B}$ is created at the base case of the recursion, where the diagonal blocks of $\mathbf{F} \cdot \mathbf{A}_1 \cdot \cdots \cdot \mathbf{A}_{\log n}$ are of size 1×1. It is shown in [40] that the kernel basis decomposition takes only $\tilde{O}(dn^{\omega})$ time to compute.

We still need to compute Equation (3). From the above algorithm, we can see that each $\mathbf{A}_i$ is a block-diagonal matrix, which consists of 2^{i-1} blocks of size $(n/2^{i-1}) \times (n/2^{i-1})$. Now we *assume* that each entry in $\mathbf{A}_i$ also has degree at most $d \cdot 2^{i-1}$. (In reality, the behavior of degrees in $\mathbf{A}_i$ may be complicated, and we need the notion of *shifted column degree* to control them; see the full version of this paper for more details.)

To compute Equation (3), we define $\mathbf{M}_i = \mathbf{A}_1\mathbf{A}_2 \ldots \mathbf{A}_i$, and compute each $\mathbf{M}_i$ by the formula

$$\mathbf{M}_{i+1} = \mathbf{M}_i\mathbf{A}_{i+1}. \tag{4}$$

The degree of each entry in $\mathbf{M}_i$ will be at most $O(2^i \cdot d)$. As we only need the results modulo x^r, we can assume the degrees are actually $O\big(\min\{r, 2^i \cdot d\}\big)$. Note that $\mathbf{A}_{i+1}$ consists of 2^i blocks, each of size $(n/2^i) \times (n/2^i)$, and the degree of each (nonempty) entry in $\mathbf{A}_{i+1}$ is also $O\big(\min\{r, 2^i \cdot d\}\big)$. Therefore, we can compute Equation (4) in

$$O\big(\min\{r, 2^i \cdot d\}\big) \cdot 2^i \cdot \mathsf{MM}(n, n/2^i, n/2^i) \tag{5}$$

time. (It is basically 2^i matrix products of size $n \times (n/2^i)$ and $(n/2^i) \times (n/2^i)$; we need to multiply another factor of $\min\{r, 2^i \cdot d\}$ which is the degree of polynomials in these matrices.)

Now, it is easy to see that the bottleneck of this algorithm occurs when $r = 2^i \cdot d$, and the time for computing Equation (4) is:

$$(5) = (r^2/d) \cdot \mathsf{MM}(n, nd/r, nd/r).$$

5 Computing Unique Shortest Paths in Directed Graphs

In this section, we show how to compute *unique* shortest paths in a directed graph in $\tilde{O}(n^{2+\mu}M)$ time, matching the current best time bound for computing the all-pairs distances [42]. Here $\mu < 0.5286$ is the solution of $\omega(1, 1, \mu) = 1 + 2\mu$ [15]. This algorithm is needed before we use Lemma 3.7.

[3] Every diagonal element of $\mathbf{B}$ is a divisor of the *largest invariant factor* of $\mathbf{F}$ (see [40, Section 5.1]), which is (again) a divisor of $\det(\mathbf{F})$. Since $\det(\mathbf{F})$ is invertible modulo x^r, every diagonal element of $\mathbf{B}$ is also invertible modulo x^r.

We may assume that before we proceed, we have already computed the all-pairs distances $\|uv\|$ for every $u, v \in V$, using the APSP algorithm in [42].

Our tie-breaking method requires a (random) permutation π of all vertices, or equivalently a bijection between the vertex set V and $[n]$, i.e. $\pi : V \to [n]$. According to π, for every graph G on V and every $u, v \in V$, we will specify a shortest path $\rho_G(u, v)$ in G from u to v in a certain way. These shortest paths will be *consistent* and *easy to compute*, which is captured by the following theorem. (See also [22, Theorem 1.3 and 1.4].)

▶ **Theorem 5.1.** *Given a graph G on V, a representation of the set of shortest paths $\{\rho_G(u, v)\}_{u,v \in V}$ can be computed in $\tilde{O}(n^{2+\mu}M)$ time, with high probability over the random choice of permutation π, such that the following hold.*

(Property a) *Let G be a graph on V. For every $u', v' \in \rho_G(u, v)$ such that u' appears before v', the portion of $u' \rightsquigarrow v'$ in $\rho_G(u, v)$ coincides with the path $\rho_G(u', v')$.*

(Property b) *Let G be a graph on V, $u, v \in V$, and G' be a subgraph of G. Suppose $\rho_G(u, v)$ is completely contained in G', then $\rho_{G'}(u, v) = \rho_G(u, v)$.*

From (Property a), for every vertex u, the shortest paths from u to every other vertex in G form a tree, and we call this tree the *outgoing shortest path tree* rooted at u, denoted as $T^{\mathsf{out}}(u)$. Similarly, the shortest paths to u from every other vertex in G also form a tree, and we call this tree the *incoming shortest path tree* rooted at u, denoted as $T^{\mathsf{in}}(u)$. Actually, the "representation" computed is exactly the set of n outgoing shortest path trees $\{T^{\mathsf{out}}(u)\}_{u \in V}$ and the set of n incoming shortest path trees $\{T^{\mathsf{in}}(u)\}_{u \in V}$.

The rest of this section

We first define the paths $\rho_G(u, v)$ in Section 5.1. Then we explain how to compute them efficiently in Section 5.2, by presenting an algorithm that computes the incoming and outgoing shortest path trees in $\tilde{O}(Mn^{2+\mu})$ time. Finally, we prove (Property a) and (Property b) in Section 5.3.

5.1 Defining $\rho_G(u, v)$

Let G be an input graph, and $\pi : V \to [n]$ be a (random) bijection. Let $u, v \in V$, P be a path from u to v, we will say that any vertex on P that is neither u nor v is an *internal vertex of P*.

Recall that we defined $|uv|$ as the *largest* number of edges in any shortest path from u to v. In particular:

- $|uv| = 0$ if and only if $u = v$;
- $|uv| = 1$ if and only if the edge (u, v) is the *only* shortest path from u to v;
- $|uv| = \infty$ if there is no path from u to v in G;
- otherwise, we have $2 \le |uv| < \infty$.

We claim that the set of vertices mapped to small values by π is a good "hitting set" w.h.p:

▷ Claim 5.2. Fix the graph G. For some large constant C, with high probability over the choice of π, the following holds. For every pair of vertices $u, v \in V$ such that $2 \le |uv| < \infty$, there is a shortest path $\rho'(u, v)$ from u to v, and an internal vertex z on $\rho'(u, v)$, such that $\pi(z) \le CMn \ln n / \|uv\|$.

Proof. Fix two vertices $u, v \in V$, and any shortest path $\rho'(u, v)$ from u to v. Denote $r = \|uv\|$, if $r \leq M \ln n$ then the claim is trivial. Otherwise, there are at least $r/1.1M$ vertices on $\rho'(u, v)$. Therefore, the probability over a random bijection $\pi : V \to [n]$ that π maps every vertex on $\rho'(u, v)$ to an integer greater than $CMn \ln n/r$ is at most

$$(1 - CM \ln n/r)^{r/1.1M} \leq 1/n^{C/1.1}.$$

Thus by a union bound, the probability that the above condition holds (for every u, v) is at least $1 - 1/n^{C/1.1-2}$, which is a high probability. ◁

Let $u, v \in V$ such that $2 \leq |uv| < \infty$. Define $w(u, v)$ as the intermediate vertex with the smallest label in any shortest path from u to v, i.e.

$$w(u, v) = \arg_w \min\{\pi(w) : \|uv\| = \|uw\| + \|wv\|, w \neq u \text{ and } w \neq v\}. \tag{6}$$

Claim 5.2 states that w.h.p. for every vertices $u, v \in V$ such that $2 \leq |uv| < \infty$, we have that

$$\pi(w(u, v)) \leq CMn \ln n/\|uv\|. \tag{7}$$

In the rest of this section, we assume that Equation (7) holds for every vertices $u, v \in V$ such that $2 \leq |uv| < \infty$. Now we define the paths $\rho_G(u, v)$.

▶ **Definition 5.3.** *Let $u, v \in V$ such that $|uv| \neq \infty$. The path $\rho_G(u, v)$ is recursively defined as follows.*

- *If $u = v$, then $\rho_G(u, v)$ is the empty path that starts and ends at u.*
- *If $|uv| = 1$, then $\rho_G(u, v)$ consists of a single edge, i.e. the edge from u to v.*
- *Otherwise, let $w = w(u, v)$, then $\rho_G(u, v)$ is the concatenation of $\rho_G(u, w)$ and $\rho_G(w, v)$.*

For every u, v such that $2 \leq |uv| < \infty$, since w is an intermediate vertex on some shortest path from u to v, it is easy to see that $|uw| < |uv|$ and $|wv| < |uv|$. Therefore $\rho_G(u, v)$ is well defined – it is inductively defined in the nondecreasing order of $|uv|$.

5.2 Computing Shortest Path Trees in $\tilde{O}(Mn^{2+\mu})$ Time

We will need the following classical algorithm for computing distance products:

▶ **Lemma 5.4** ([42]). *Let A be an $n \times m$ matrix, and B be an $m \times n$ matrix. Suppose every entry in A or B is either $+\infty$ or an integer with absolute value at most M. Then the distance product of A and B can be computed in $\tilde{O}(M \cdot \mathsf{MM}(n, m, n))$ time.*

Computing $w(u, v)$

We first show how to compute $w(u, v)$ for every $u, v \in V$ such that $2 \leq |uv| < \infty$ in $\tilde{O}(Mn^{2+\mu})$ time. Then we use the values of all $w(u, v)$ to compute the incoming and outgoing shortest path trees in $\tilde{O}(n^2)$ additional time. Our strategy for computing $w(u, v)$ is to mimic the algorithm in [17, 28] for computing maximum witness of Boolean matrix multiplication. In particular, we divide the possible witnesses into blocks, and use fast matrix multiplication algorithms to find the block containing $w(u, v)$, for every u, v. After that, we use brute force to find $w(u, v)$ inside that block. Details follow.

Let $r = 2^k$ be a parameter, we show how to compute $w(u, v)$ for every pair of vertices $u, v \in V$ such that $r \leq \|uv\| < 2r$. Let

$$\mathcal{H}_r = \{z \in V : \pi(z) \le CMn \ln n/r\}.$$

By Claim 5.2, for every vertices u, v such that $\|uv\| \in [r, 2r)$, we have $w(u, v) \in \mathcal{H}_r$.

We define an $n \times |\mathcal{H}_r|$ matrix A and an $|\mathcal{H}_r| \times n$ matrix B as follows. For every $u \in V$ and $z \in \mathcal{H}_r$, we define

$$A[u, z] = \begin{cases} \|uz\| & \text{if } \|uz\| \le 2r \text{ and } u \ne z \\ +\infty & \text{otherwise} \end{cases}, \text{ and } B[z, u] = \begin{cases} \|zu\| & \text{if } \|zu\| \le 2r \text{ and } u \ne z \\ +\infty & \text{otherwise} \end{cases}.$$

Then we compute the *minimum witness* of the distance product $A \star B$. To be more precise, we compute the matrix $W[\cdot, \cdot]$ such that for every $u, v \in V$,

$$W[u, v] = \arg_z \min\{\pi(z) : \|uv\| = A[u, z] + B[z, v]\}.$$

Correctness. Fix $u, v \in V$, where $\|uv\| \in [r, 2r)$. We will show that if $|uv| = 1$, then $W[u, v]$ does not exist; otherwise $W[u, v]$ coincides with $w(u, v)$ defined in Equation (6).

First, suppose $|uv| = 1$, then there are no intermediate vertex z such that $\|uv\| = \|uz\| + \|zv\|$, which means $W[u, v]$ does not exist.

Now we assume $|uv| \ge 2$. Since $\|uv\| \ge r$, by Claim 5.2, there is an intermediate vertex $z \in \mathcal{H}_r$ such that $\|uz\| + \|zv\| = \|uv\|$. Since $\|uz\|, \|zv\| \le \|uv\| < 2r$, we can see that $\|uv\| = A[u, z] + B[z, v]$, therefore $W[u, v]$ exists. Let $z = W[u, v]$, then by Equation (6), $\pi(w(u, v)) \le \pi(z)$. On the other hand, Claim 5.2 shows that $w(u, v) \in \mathcal{H}_r$, so by the definition of $z = W[u, v]$, we have $\pi(z) \le \pi(w(u, v))$. Therefore $z = w(u, v)$ and we have established the correctness of $W[\cdot, \cdot]$.

Time complexity. Now we show how to compute the matrix $W[\cdot, \cdot]$ efficiently.

Let $s = n^\mu$, where $\mu \in (0, 1)$ is a parameter to be determined later. If $|\mathcal{H}_r| < s$, then we can compute the matrix W by brute force in $\tilde{O}(n^2 s)$ time. Otherwise, we partition $\mathcal{H}_r$ into blocks of size s, where the i-th block contains vertices that are mapped by π to values between $(i-1) \cdot s + 1$ and $i \cdot s$. For every block i, we compute the distance product of A and B where only vertices in block i are allowed as witnesses. In other words, we compute the following matrix

$$D^i[u, v] = \min\{A[u, z] + B[z, v] : (i-1) \cdot s + 1 \le \pi(z) \le i \cdot s\}.$$

By Lemma 5.4, this matrix can be computed in $\tilde{O}(r \cdot \mathsf{MM}(n, s, n))$ time. There are $O(|\mathcal{H}_r|/s) = \tilde{O}(Mn/(rs))$ blocks, and we need to compute a distance product D^i for each block i. Therefore the total time for computing all these distance products is

$$\tilde{O}(r \cdot \mathsf{MM}(n, s, n) \cdot Mn/(rs)) = \tilde{O}(M \cdot (n/s) \cdot \mathsf{MM}(n, s, n)).$$

Now for every $u, v \in V$ such that $\|uv\| \in [r, 2r)$ and $|uv| \ge 2$, we want to compute $W[u, v]$, which is the vertex $z \in \mathcal{H}_r$ with the minimum $\pi(z)$, such that $\|uv\| = A[u, z] + B[z, v]$. First, we find the smallest i such that $D^i[u, v] = \|uv\|$, and we know that $W[u, v]$ is in the i-th block. (If such i does not exist, then $W[u, v]$ does not exist either, and $|uv| = 1$.) This step takes $\tilde{O}(Mn/(rs))$ time. Then we iterate through the vertices in this block, and find the vertex z with the smallest $\pi(z)$ such that $A[u, z] + B[z, v] = \|uv\|$. This step takes $O(s)$ time.

It follows that the time complexity for computing every $w(u,v)$ where $\|uv\| \in [r, 2r)$ is

$$\begin{aligned}&\tilde{O}(M \cdot \mathsf{MM}(n,s,n) \cdot (n/s) + n^2 \cdot Mn/(rs) + n^2 s)\\ \leq\ &\tilde{O}(M \cdot \mathsf{MM}(n,s,n) \cdot (n/s) + n^2 s) \qquad (8)\\ \leq\ &\tilde{O}(M \cdot n^{\omega(1,\mu,1)+1-\mu} + n^{2+\mu}).\end{aligned}$$

Here, Equation (8) is because $n^2 \cdot Mn/(rs) \leq n^2 \cdot M \cdot (n/s) \leq M \cdot \mathsf{MM}(n,s,n) \cdot (n/s)$.

Let μ be the solution to $\omega(1,\mu,1) = 1 + 2\mu$, then $\mu < 0.5286$ ([15,42]). It follows that the time complexity for computing every $w(u,v)$, where $r \leq \|uv\| < 2r$, is at most $\tilde{O}(Mn^{2+\mu})$.

Putting it together. We run the above algorithm for k from 0 to $\lfloor \log(nW) \rfloor$, and for each k, we update the values $w(u,v)$ where $\|uv\| \in [2^k, 2^{k+1})$. The total time to compute $w(u,v)$ for all u, v is thus $\tilde{O}(Mn^{2+\mu})$.

From $w(u,v)$ to unique shortest paths

For every $u, v \in V$, we will compute the parent of u in the tree $T^{\mathsf{in}}(v)$, denoted as $\mathsf{parent}_v(u)$. In other words, $\mathsf{parent}_v(u)$ is the second vertex in the path $\rho_G(u,v)$ (the first being u). After computing $\mathsf{parent}_v(u)$ for every $u, v \in V$, it is easy to construct $T^{\mathsf{in}}(v)$ for every vertex v. We can compute every $T^{\mathsf{out}}(u)$ in a symmetric fashion.

We proceed by nondecreasing order of $\|uv\|$. Suppose that for every (u', v') such that $\|u'v'\| < \|uv\|$, we have already computed $\mathsf{parent}_{v'}(u')$. Now we compute $\mathsf{parent}_v(u)$ as follows. Let $w = w(u,v)$. If w does not exist, let $\mathsf{parent}_v(u) = v$; otherwise $\mathsf{parent}_v(u) = \mathsf{parent}_w(u)$.

This algorithm (that given every $w(u,v)$, computes every $\mathsf{parent}_v(u)$) clearly runs in $\tilde{O}(n^2)$ time. Notice that if w exists, then w is an intermediate vertex in $\rho_G(u,v)$, thus $\|uw\| < \|uv\|$, and the second vertex in the path $\rho_G(u,v)$ coincides with the second vertex in the path $\rho_G(u,w)$. Hence, the correctness of the algorithm can be easily proved by induction on $\|uv\|$.

5.3 Proof of Theorem 5.1

▶ **Theorem 5.1.** *Given a graph G on V, a representation of the set of shortest paths $\{\rho_G(u,v)\}_{u,v\in V}$ can be computed in $\tilde{O}(n^{2+\mu}M)$ time, with high probability over the random choice of permutation π, such that the following hold.*

(Property a) *Let G be a graph on V. For every $u', v' \in \rho_G(u,v)$ such that u' appears before v', the portion of $u' \rightsquigarrow v'$ in $\rho_G(u,v)$ coincides with the path $\rho_G(u',v')$.*

(Property b) *Let G be a graph on V, $u, v \in V$, and G' be a subgraph of G. Suppose $\rho_G(u,v)$ is completely contained in G', then $\rho_{G'}(u,v) = \rho_G(u,v)$.*

In this subsection, for any path P and vertices $u', v' \in P$ such that u' appears before v' on P, we use $P[u', v']$ to denote the portion of $u' \rightsquigarrow v'$ on the path P.

Proof of (Property a). We prove it by induction on the number of edges of $\rho_G(u,v)$. Let $P = \rho_G(u,v)$. If $u = v$ or P has only one edge, (Property a) is trivial. Now suppose P has k edges where $k > 1$. Let $w = w(u,v)$, then w must lie on P. Consider the following three cases:

- Suppose u' appears after (or coincides with) w on P. By definition, $P[w,v] = \rho_G(w,v)$. Then $P[u',v'] = \rho_G(u',v')$ by induction hypothesis on $\rho_G(w,v)$ since it has fewer edges than $\rho_G(u,v)$.
- Suppose v' appears before (or coincides with) w. This case is symmetric to the above case.

- Otherwise, w lies between u' and v' on P.

  First, we claim that $w = w(u', v')$. As w lies on some shortest path from u' to v' (i.e. $P[u', v']$), we have $\pi(w(u', v')) \leq \pi(w)$. On the other hand, suppose there exists w' such that $\pi(w') < \pi(w)$ and w' is on some shortest path from u' to v'. Then w' also lies on some shortest path from u to v, so it is a better candidate for $w(u, v)$, contradicting the definition of w.

  Second, by induction hypothesis on $\rho_G(u, w)$, which has fewer edges than $\rho_G(u, v)$, we have $P[u', w] = \rho_G(u', w)$. Similarly, $P[w, v'] = \rho_G(w, v')$. Therefore, by definition, $P[u', v'] = P[u', w] \circ P[w, v'] = \rho_G(u', v')$. ◀

Proof of (Property b). We prove it by induction on the number of edges of $\rho_G(u, v)$. Let $P = \rho_G(u, v)$. If $u = v$ or P has only one edge, (Property b) is trivial.

Now suppose P has more than one edge. Let $w = w_G(u, v)$ (i.e. the vertex $w(u, v)$ defined in Equation (6) in graph G), we claim that w coincides with $w_{G'}(u, v)$ (i.e. the vertex $w(u, v)$ defined in Equation (6) in graph G'). Since P is also a shortest path from u to v in G', we have $\pi(w_{G'}(u, v)) \leq \pi(w)$. On the other hand, suppose there exists w' such that $\pi(w') < \pi(w)$ and w' is on some shortest path from u to v in G'. Then w' also lies on some shortest path from u to v in G, so it is a better candidate for $w_G(u, v)$, contradicting the definition of w.

Since $\rho_G(u, w)$ has fewer edges than $\rho_G(u, v)$, and $\rho_G(u, w)$ is completely contained in G', we can use induction hypothesis on $\rho_G(u, w)$ to conclude that $P[u, w] = \rho_{G'}(u, w)$. Similarly, we can use the induction hypothesis on $\rho_G(w, v)$ to conclude that $P[w, v] = \rho_{G'}(w, v)$. Therefore, by definition, $\rho_{G'}(u, v) = \rho_{G'}(u, w) \circ \rho_{G'}(w, v) = P$. ◀

6 Conclusions and Open Problems

We presented an improved DSO for directed graphs with integer weights in $[1, M]$. The preprocessing time is $O(n^{2.5794}M)$ and the query time is $O(1)$. However, there is still a small gap between the preprocessing time of our DSO and the current best time bound for the APSP problem in directed graphs, which is $\tilde{O}(n^{2+\mu}M) \leq O(n^{2.5286}M)$ [42]. Can we improve the preprocessing time to $\tilde{O}(n^{2+\mu}M)$, matching the latter time bound? Another interesting problem is to investigate the complexity of preprocessing a DSO in undirected graphs – here, the best time bound for APSP is $\tilde{O}(n^{\omega}M)$ [27, 30]. Can we preprocess a DSO in $\tilde{O}(n^{\omega}M)$ time on undirected graphs?

Compared to other DSOs [8, 16, 36], our oracle has two drawbacks. First, our query algorithm only outputs the shortest distance, but we do not know how to find the actual shortest paths. So another open problem is whether we can find the actual shortest path with additional $O(l)$ query time, where l is the number of edges in the returned shortest path. Second, since we used [22, Observation 2.1], our oracle can only deal with positive edge weights. Can we extend our oracle to also deal with negative edge weights?

For every parameter f, the r-truncated DSO in Section 3.2 can actually handle f edge/vertex deletions in $\tilde{O}(f^{\omega}r)$ query time. (See also [35].) However, as far as we know, [22, Observation 2.1] only works for one failure. It would be exciting to extend [22, Observation 2.1] or our (full) DSO to also handle f failures.

References

1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

2 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proc. 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539, 2021. doi:10.1137/1.9781611976465.32.

3 Noga Alon, Shiri Chechik, and Sarel Cohen. Deterministic combinatorial replacement paths and distance sensitivity oracles. In *Proc. 46th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 132 of *LIPIcs*, pages 12:1–12:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.12.

4 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997. doi:10.1006/jcss.1997.1388.

5 Aaron Bernstein and David R. Karger. Improved distance sensitivity oracles via random sampling. In *Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 34–43, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347087.

6 Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *Proc. 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 101–110, 2009. doi:10.1145/1536414.1536431.

7 James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974. doi:10.2307/2005828.

8 Shiri Chechik and Sarel Cohen. Distance sensitivity oracles with subcubic preprocessing time and fast query time. In *Proc. 52nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 1375–1388, 2020. doi:10.1145/3357713.3384253.

9 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

10 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, 51(6):968–992, 2004. doi:10.1145/1039488.1039492.

11 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM Journal of Computing*, 37(5):1299–1318, 2008. doi:10.1137/S0097539705429847.

12 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 506–515, 2009. doi:10.1137/1.9781611973068.56.

13 Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384–391, 2009. doi:10.1137/1.9781611973068.43.

14 Ran Duan and Tianyi Zhang. Improved distance sensitivity oracles via tree partitioning. In *Proc. 15th International Symposium on Algorithms and Data Structures (WADS)*, volume 10389 of *LNCS*, pages 349–360, 2017. doi:10.1007/978-3-319-62127-2_30.

15 Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1029–1046, 2018. doi:10.1137/1.9781611975031.67.

16 Fabrizio Grandoni and Virginia Vassilevska Williams. Faster replacement paths and distance sensitivity oracles. *ACM Transactions on Algorithms*, 16(1):15:1–15:25, 2020. doi:10.1145/3365835.

17 Miroslaw Kowaluk and Andrzej Lingas. LCA queries in directed acyclic graphs. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *LNCS*, pages 241–248, 2005. doi:10.1007/11523468_20.

18 George Labahn, Vincent Neiger, and Wei Zhou. Fast, deterministic computation of the Hermite normal form and determinant of a polynomial matrix. *Journal of Complexity*, 42:44–71, 2017. doi:10.1016/j.jco.2017.03.003.

19 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. 39th International Symposium on Symbolic and Algebraic Computation, (ISSAC)*, pages 296–303, 2014. doi:10.1145/2608628.2608664.

20 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1236–1252, 2018. doi:10.1137/1.9781611975031.80.

21 Grazia Lotti and Francesco Romani. On the asymptotic complexity of rectangular matrix multiplication. *Theoretical Computer Science*, 23:171–185, 1983. doi:10.1016/0304-3975(83)90054-3.

22 Hanlin Ren. Improved distance sensitivity oracles with subcubic preprocessing time. In *Proc. 28th European Symposium on Algorithms (ESA)*, volume 173 of *LIPIcs*, pages 79:1–79:13, 2020. doi:10.4230/LIPIcs.ESA.2020.79.

23 Karthik C. S. and Merav Parter. Deterministic replacement path covering. In *Proc. 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 704–723, 2021. doi:10.1137/1.9781611976465.44.

24 Piotr Sankowski. Shortest paths in matrix multiplication time. In *Proc. 13th European Symposium on Algorithms (ESA)*, volume 3669 of *LNCS*, pages 770–778, 2005. doi:10.1007/11561071_68.

25 Piotr Sankowski. Subquadratic algorithm for dynamic shortest distances. In *Proc. 11th International Computing and Combinatorics Conference (COCOON)*, volume 3595 of *LNCS*, pages 461–470, 2005. doi:10.1007/11533719_47.

26 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

27 Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995. doi:10.1006/jcss.1995.1078.

28 Asaf Shapira, Raphael Yuster, and Uri Zwick. All-pairs bottleneck paths in vertex weighted graphs. *Algorithmica*, 59(4):621–633, 2011. doi:10.1007/s00453-009-9328-x.

29 Jack Sherman and Winifred J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950. URL: http://www.jstor.org/stable/2236561.

30 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proc. 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 605–615, 1999. doi:10.1109/SFFCS.1999.814635.

31 Arne Storjohann. High-order lifting and integrality certification. *Journal of Symbolic Computation*, 36(3-4):613–648, 2003. doi:10.1016/S0747-7171(03)00097-X.

32 Andrew James Stothers. *On the complexity of matrix multiplication*. PhD thesis, The University of Edinburgh, 2010.

33 Jan van den Brand and Danupon Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 436–455, 2019. doi:10.1109/FOCS.2019.00035.

34 Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 456–480, 2019. doi:10.1109/FOCS.2019.00036.

35 Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 424–435, 2019. doi:10.1109/FOCS.2019.00034.

36 Oren Weimann and Raphael Yuster. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Transactions on Algorithms*, 9(2):14:1–14:13, 2013. doi:10.1145/2438645.2438646.

37 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 887–898, 2012. doi:10.1145/2213977.2214056.

38 Max A Woodbury. Inverting modified matrices. *Memorandum report*, 42(106):336, 1950.

39 Wei Zhou, George Labahn, and Arne Storjohann. Computing minimal nullspace bases. In *Proc. 37th International Symposium on Symbolic and Algebraic Computation, (ISSAC)*, pages 366–373, 2012. doi:10.1145/2442829.2442881.

40 Wei Zhou, George Labahn, and Arne Storjohann. A deterministic algorithm for inverting a polynomial matrix. *Journal of Complexity*, 31(2):162–173, 2015. doi:10.1016/j.jco.2014.09.004.

41 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79*, volume 72 of *LNCS*, pages 216–226, 1979. doi:10.1007/3-540-09519-5_73.

42 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.