2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)

Sensitive Distance and Reachability Oracles for Large Batch Updates

Jan van den Brand
KTH Royal Institute of Technology
Stockholm, Sweden

Thatchaphol Saranurak
Toyota Technological Institute at Chicago
Chicago, USA

***Abstract*—In the *sensitive distance oracle* problem, there are three phases. We first preprocess a given *directed* graph G with n nodes and integer weights from $[-W, W]$. Second, given a single batch of f edge insertions and deletions, we update the data structure. Third, given a query pair of nodes (u, v), return the distance from u to v. In the easier problem called *sensitive reachability oracle* problem, we only ask if there exists a directed path from u to v.**

Our first result is a sensitive distance oracle with $\tilde{O}(Wn^{\omega+(3-\omega)\mu})$ preprocessing time, $\tilde{O}(Wn^{2-\mu}f^2 + Wnf^{\omega})$ update time, and $\tilde{O}(Wn^{2-\mu}f + Wnf^2)$ query time where the parameter $\mu \in [0, 1]$ can be chosen. The data-structure requires $O(Wn^{2+\mu}\log n)$ bits of memory. This is the first algorithm that can handle $f \geq \log n$ updates. Previous results (e.g. [Demetrescu et al. SICOMP'08; Bernstein and Karger SODA'08 and FOCS'09; Duan and Pettie SODA'09; Grandoni and Williams FOCS'12]) can handle at most 2 updates. When $3 \leq f \leq \log n$, the only non-trivial algorithm was by [Weimann and Yuster FOCS'10]. When $W = \tilde{O}(1)$, our algorithm simultaneously improves their preprocessing time, update time, and query time. In particular, when $f = \omega(1)$, their update and query time is $\Omega(n^{2-o(1)})$, while our update and query time are *truly subquadratic* in n, i.e., ours is faster by a polynomial factor of n. To highlight the technique, ours is the first graph algorithm that exploits the *kernel basis decomposition* of polynomial matrices by [Jeannerod and Villard J.Comp'05; Zhou, Labahn and Storjohann J.Comp'15] developed in the symbolic computation community.

As an easy observation from our technique, we obtain the first sensitive reachability oracle can handle $f \geq \log n$ updates. Our algorithm has $O(n^{\omega})$ preprocessing time, $O(f^{\omega})$ update time, and $O(f^2)$ query time. This data-structure requires $O(n^2 \log n)$ bits of memory. Efficient sensitive reachability oracles were asked in [Chechik, Cohen, Fiat, and Kaplan SODA'17]. Our algorithm can handle any constant number of updates in constant time. Previous algorithms with constant update and query time can handle only at most $f \leq 2$ updates. Otherwise, there are non-trivial results for $f \leq \log n$, though, with query time $\Omega(n)$ by adapting [Baswana, Choudhary and Roditty STOC'16].

***Keywords*-sensitive oracle; emergency oracle; failure; batch; reachability; distance;**

I. Introduction

In the *sensitive distance oracle* problem[1], there are three phases. First, we preprocess a given *directed* graph G with n nodes, m edges, and integer weights from $[-W, W]$. Second, given a single batch of f edge insertions and deletions, we update the data structure. Third, given a pair of nodes (u, v), return the distance from u to v. The time taken in each phase is called *preprocessing time*, *update time*, and *query time* respectively. In an easier problem called *sensitive reachability oracle* problem, the setting is the same except that we only ask if there exists a directed path from u to v.

Although both problems are well-studied (as will be discussed below), all existing non-trivial algorithms only handle $f \leq \log n$ updates. In contrast, in the analogous problems in undirected graphs, algorithms that handle updates of any size are known. For example, there are sensitive connectivity oracles [1], [2], [3], [4] (i.e. reachability oracles in which is undirected graphs) with $\mathrm{poly}(n)$ preprocessing time and $\mathrm{poly}(f, \log n)$ update and query time for *every* f. For sensitive distance oracles in undirected graphs (studied in [5], [6], [7]), Chechik, Langberg, Peleg, and Roditty [6] show that this is possible as well if *approximate* distance is allowed. It is interesting whether there is any inherent barrier for $f \geq \log n$ in directed graphs. In this paper, we show that there is no such barrier.

Sensitive distance oracle.: All previous works on this problem handle edge *deletions* only[2]. The first result is in the case when $f = 1$ by Demetrescu et al. [8][3] since 2002. Their algorithm has $O(mn^{1.5})$ preprocessing time and $O(1)$ update and query time. This preprocessing time is improved by Bernstein and Karger to $\tilde{O}(n^2\sqrt{m})$ [9] and finally to $\tilde{O}(mn)$ [10]. The $\tilde{O}(mn)$ bound is optimal up to a sub-polynomial factor unless there is a truly subcubic algorithm for the all-pairs shortest path problem. Later, Grandoni and Williams [11] showed a new trade-off. For example, they can obtain $O(Wn^{2.88})$ preprocessing time and $O(n^{0.7})$ update and query time.

It was stated as an open question in [8], [9], [10] whether there is a non-trivial algorithm for handling more than one deletion. Duan and Pettie [12] give an affirmative answer for $f = 2$. Their algorithm has $\mathrm{poly}(n)$ preprocessing time and $\tilde{O}(1)$ update and query time. They however stated that "it is practically infeasible" to extend their algorithm to handle three deletions. Weimann and Yuster [13] later showed an algorithm for handling up to $f \leq \log n / \log\log n$

[1]In literature this setting is also referred to as *distance sensitivity oracle* or *emergency algorithm*.

[2]Node deletions can be handled as well using a simple reduction.

[3]The result was published in 2008, but previously announced at SODA and ISAAC 2002.

2575-8454/19/$31.00 ©2019 IEEE
DOI 10.1109/FOCS.2019.00034

deletions. For any parameter $\mu \in [0,1]$, their algorithm has $\tilde{O}(Wn^{\omega+\mu})$ preprocessing time and $\tilde{O}(n^{2-\mu/f})$ update and query time. This algorithm remains the only non-trivial algorithm when $f \geq 3$.

To summarize, no previous algorithm can handle $f > \log n/\log\log n$ updates. Another drawback of all previous algorithms is that they inherently cannot handle changes of edge weights.[4] In the setting where we allow changes of edge weights, it is quite natural to ask for algorithms that handle somewhat large batches of updates.[5]

In this work, we show the first sensitive distance oracle that can handle *any* number of updates. Moreover, we allow both edge insertions, deletions, and changes of weights. Ours is the fastest oracle for handling $f \geq 3$ updates when $W = \tilde{O}(1)$:

Theorem 1.1. *For any parameter $\mu \in [0,1]$, there is a Monte Carlo data structure that works in three phases as follows:*

1) *Preprocess a directed graph with n nodes and integer weights from $[-W,W]$ in $\tilde{O}(Wn^{\omega+(3-\omega)\mu})$ time and store $O(Wn^{2+\mu}\log n)$ bits.*
2) *Given updates to any set of f edges, update the data structure in $\tilde{O}(Wn^{2-\mu}f^2 + Wnf^{\omega})$ time and store additional $O(Wnf^2\log n)$ bits.*
3) *Given a pair of nodes (u,v), return the distance from u to v or report that there is a negative cycle in $\tilde{O}(Wn^{2-\mu}f + Wnf^2)$ time.*

The algorithm is Monte Carlo randomized and is correct with high probability, i.e. the algorithm may return a larger distance with probability at most $1/n^c$ for any constant c. We note that, any algorithm that can handle edge deletions can also handle node deletions by using a standard reduction.

Let us compare this result to the algorithm by Weimann and Yuster [13]. First, instead of edge deletions only, we allow *both* edge insertions, deletions, and changes of weight. The second point is efficiency. For any $\mu \in [0,1]$, we improve the preprocessing time from $\tilde{O}(Wn^{\omega+\mu})$ to $\tilde{O}(Wn^{\omega+(3-\omega)\mu})$. Both of our update and query time are bounded above by $\tilde{O}(Wn^{2-\mu}f^{\omega})$. When $W = \tilde{O}(1)$, this improves their bound of $\tilde{O}(n^{2-\mu/f})$ for any $f \geq 2$ by a polynomial factor. In particular, when $\mu > 0$ and $f = \omega(1)$, their bound is $\Omega(n^{2-o(1)})$, while ours is *truly subquadratic* in n. The last and most conceptually important point is that we remove the constraint that $f \leq \log n/\log\log n$. While their technique is inherently for small f, our approach is different and can handle any number of updates.

To highlight our technique, this is the first graph algorithm that exploits the powerful *kernel basis decomposition* of polynomial matrices by Jeannerod and Villard [14] and Zhou, Labahn and Storjohann [15] from the symbolic computation community. We explain the overview of our algebraic techniques in Section I-A. We then compare our techniques to the previous related works in Section I-B.

Sensitive reachability oracle.: It was asked as an open problem by Chechik et al. [7] whether there is a much more efficient data structure if the query is only about reachability between nodes and not distance. As an easy observation from our technique, we give a strong affirmative answer to this question:

Theorem 1.2. *There is a Monte Carlo sensitive reachability oracle that preprocess an n-node graph in $O(n^{\omega})$ time and stores $O(n^2\log n)$ bits. Then, given a set of f edge insertions/deletions and node deletions, update the data structure in $O(f^{\omega})$ time and store additional $O(f^2\log n)$ bits. Then, given a query (u,v), return if there is directed path from u to v in $O(f^2)$ time.*

Previously, there are only algorithms that handle $f = 1$ edge deletions based on dominator trees [16], [17], [18], [19] or $f \leq 2$ edge deletions [20]. These algorithms have $O(1)$ update and query time. Another approach based on fault-tolerant subgraphs[6] gives algorithms with query time at least $\Omega(n)$ [21], [22] and becomes trivial when $f \geq \log n$. When $f = O(1)$, our algorithm has $O(1)$ update and query time like the algorithms using the first approach. Moreover, ours is the first which handles updates of any size.

It was shown in [23], [24] that, assuming the *Boolean Matrix Multiplication* conjecture, there cannot be any constant $\epsilon > 0$ and a "combinatorial" algorithm for Theorem 1.2 which has $O(n^{3-\epsilon})$ preprocessing time, and can handle 2 edge insertions using $O(n^{2-\epsilon})$ update and query time. Our result does not refute the conjecture as we use fast matrix multiplication.

A. Technical overview

Set the stage.: The first step of all our results is to reduce the problem on graphs to algebraic problems using the following known reduction (see Lemma 2.1 for a more detailed statement). Let G be an n-node graph with integer weights from $[-W,W]$. Let $\mathbb{F}$ be a finite field of size at least n^3. We construct a polynomial matrix $A \in \mathbb{F}[X]^{n\times n}$ such that $A_{i,j} = a_{i,j}X^{W+c_{i,j}}$ where $c_{i,j}$ is the weight of edge (i,j) and $a_{i,j}$ is a random element from $\mathbb{F}$. Then, with high probability, we can read off the distance from i to j in G from the (i,j) entry of the adjoint matrix $\mathrm{adj}(A)$ of A, for

[4] One restricted and inefficient approach for handling weight increases via edge deletions is as follows. For each edge e, we need to know all possible weights for e, say $w_1 \leq \cdots \leq w_k$. Then, we add multi-edges $e_1,\ldots,e_k$ with the same endpoints as e, and the weight of e_i is w_i. To increase the weight from w_1 to w_j, we delete *all* edges $e_1,\ldots,e_{j-1}$. However, all previous algorithms handle only small number deletions.

[5] For example, in a road network, there usually are not too many completely blocked road sections (i.e. deleted edges), but during rush-hour there can be many roads with increased congestion (i.e. edges with increased weight).

[6] The goal is to find sparse subgraphs preserving reachability information of the original graph even after some edges are deleted.

all pairs (i, j). That is, it suffices to build a data structure on the above polynomial matrix A that can handle updates and can return an entry of its adjoint $\mathrm{adj}(A)$. Updating f edges in G corresponds to adding A with C where $C \in \mathbb{F}[X]^{n\times n}$ has f non-zero entries. Further we have with high probability that $\det(A) \neq 0$, so it is enough to focus on algorithms that work on non-singular matrices. From now, we let $\mathbb{F}$ be an arbitrary field and we use the number of field operations as complexity measure.

Warm up: slow preprocessing.: To illustrate the basic idea how to maintain the adjoint, we will prove the following:

Lemma 1.3. *Let $A \in \mathbb{F}[X]^{n\times n}$ be a polynomial matrix of degree d with $\det(A) \not\equiv 0$, then there exists an algorithm that preproceses A in $\tilde{O}(dn^{\omega+1})$ operations. Then, for any $C \in \mathbb{F}[X]^{n\times n}$ with f non-zero entries of degree at most d, we can query any entry of $\mathrm{adj}(A + C)$ in $\tilde{O}(dnf^{\omega+1})$ operations, if $\det(A + C) \neq 0$.*

If $d = 0$, then the preprocessing and query time are $O(n^\omega)$ and $O(f^\omega)$ respectively.

This immediately implies a weaker statement of Theorem 1.1 when $\mu = 1$ and the edge updates and the pair to be queried are given at the same time. We remark that, from the simple proof below, this already gives us the first non-trivial sensitive distance oracle which can handle any number of f updates. Previous techniques inherently require $f \leq \log n/\log\log n$. As the reachability problem can be considered a shortest path problem, where every edge has weight zero, we also obtain a result similar to Theorem 1.2 from Lemma 1.3 for $d = 0$.

Corollary 1.4. *Let G be some directed graph with integer weights in $[-W, W]$, then there exists an algorithm that preprocess G in $\tilde{O}(Wn^{\omega+1})$ time. Then, given f edge updates to G and a query pair (u, v), it returns the distance from u to v in the updated graph in $\tilde{O}(Wnf^{\omega+1})$ time.*

Corollary 1.5. *Let G be some directed graph, then there exists an algorithm that preprocess G in $\tilde{O}(n^\omega)$ time. Then, given f edge updates to G and a query pair (u, v), it returns the reachability from u to v in the updated graph in $\tilde{O}(f^\omega)$ time.*

To prove Lemma 1.3, we use the key equality below based on the Sherman-Morrison-Woodbury formula. The proof is deferred to Section VI.

Lemma 1.6. *Let A be an $n \times n$ matrix and U, V be $n \times f$ matrices, such that $\det(A), \det(A + UV^\top) \neq 0$. Define the $f \times f$ matrix $M := \mathbb{I}\cdot\det(A) + V^\top \mathrm{adj}(A)U$. Then, we have*

$$\mathrm{adj}(A + UV^\top) = (\mathrm{adj}(A)\det(M) - (\mathrm{adj}(A)U)\ \mathrm{adj}(M)\ (V^\top \mathrm{adj}(A)))\det(A)^{-f}.$$

The algorithm for Lemma 1.3 is as follows: We preprocess A by computing $\det(A)$ and $\mathrm{adj}(A)$. As $\det(A)$ and $\mathrm{adj}(A)$ have degree at most dn, this takes $\tilde{O}(dn \times n^\omega) = \tilde{O}(dn^{\omega+1})$ field operations [25, Chapter 1] (or just $O(n^\omega)$ if $d = 0$). Next, we write $C = UV^\top$ where $U, V \in \mathbb{F}[X]^{n\times f}$ have only one non-zero entry of degree $\leq d$ per column. To compute $\mathrm{adj}(A + UV^\top)_{i,j}$, we simply compute

$$\mathrm{adj}(A)_{i,j}\frac{\det(M)}{\det(A)^f} - \Bigg(\overbrace{(\vec{e}_i^\top \mathrm{adj}(A)U)}^{:=\vec{u}} \cdot \mathrm{adj}(\overbrace{\mathbb{I}\cdot\det(A) + V^\top \mathrm{adj}(A)U}^{=:M}) \cdot \overbrace{(V^\top \mathrm{adj}(A)\vec{e}_j)}^{:=\vec{v}}\Bigg)\det(A)^{-f}$$

where $\vec{e}_i$ is the i-th standard unit vector. This computation can be separated into the following steps:

1) Compute $\mathrm{adj}(A)_{i,j}$, $\vec{u} := \vec{e}_i^\top \mathrm{adj}(A)U$ and $\vec{v} := V^\top \mathrm{adj}(A)\vec{e}_j$. Note that because of the sparsity of U and V, $\vec{u}$ and $\vec{v}$ are essentially just vectors of f elements of $\mathrm{adj}(A)$, each multiplied by a non-zero element of U and V. Thus this step can be summarized as obtaining $O(f)$ elements of $\mathrm{adj}(A)$.
2) Compute $V^\top \mathrm{adj}(A)U$ which are likewise just f^2 elements of $\mathrm{adj}(A)$, each multiplied by a non-zero element of U and V. So this time we have to obtain $O(f^2)$ entries of $\mathrm{adj}(A)$.
3) Compute adjoint $\mathrm{adj}(M)$ and determinant $\det(M)$.
4) Compute $\det(A)^f$.
5) Compute $\mathrm{adj}(A)_{i,j}\frac{\det(M)}{\det(A)^f}$ and vector-matrix-vector product $\vec{u}\,\mathrm{adj}(M)\vec{v}$ and divide it by $\det(A)^f$. Then subtract the two values and we obtain $\mathrm{adj}(A + UV^\top)_{i,j}$.

Steps 1 and 2 require only $\tilde{O}(dnf^2)$ field operations as we just have to read $O(f^2)$ entries of $\mathrm{adj}(A)$ and multiply them by some small d-degree polynomials from U and V. In step 3 we have to compute the adjoint and determinant of a $f \times f$ matrix of degree dn. This takes $\tilde{O}(dnf^{\omega+1})$ operations. Step 4 computes $\det(A)^f$ where $\det(A)$ is of degree dn, which takes $\tilde{O}(dnf)$ operations. Step 5 takes $\tilde{O}(dnf^3)$ because $\mathrm{adj}(M)$ is a degree dnf matrix of dimension $f \times f$. The total number of operations is thus $\tilde{O}(dnf^{\omega+1})$. The algorithm does not require the upper bound of $f \leq \log n/\log\log n$ as in [13].

For the reachability case, when $d = 0$, all entries of the matrices and vectors are just field elements, so steps 1 and 2 need only $O(f^2)$ operations. Step 3 needs $O(f^\omega)$ operations, while 4 can be done in just $O(\log f)$ operations, and the last step 5 requires only $O(f^2)$ operations.

Key technique: kernel basis decomposition: The biggest bottleneck in Lemma 1.3 is explicitly computing $\mathrm{adj}(A)$ and $\mathrm{adj}(M)$. For $\mathrm{adj}(A)$ it already takes $\Omega(dn^3)$ operations

in the preprocessing step just to write down the n^2 entries of $\mathrm{adj}(A)$ each of which has degree upto dn. What we need is an *adjoint oracle*, i.e. a data structure on A with fast preprocessing that can still quickly answer queries about entries of $\mathrm{adj}(A)$.

By replacing this data structure in the five steps of the proof of Lemma 1.3, this immediately gives distance oracles in the sensitive setting which dominate previous results when $W = \tilde{O}(1)$. (This is how we obtain Theorem 1.1).

The key contribution of this paper is to realize that the technique in [14], [15] actually gives the desired adjoint oracle. This technique, which we call the *kernel basis decomposition*, is introduced by Jeannerod and Villard [14] and then improved by Zhou, Labahn, and Storjohann [15]. It is originally used for inverting a polynomial matrix of degree d in $\tilde{O}(dn^3)$ operations. However, the following adjoint oracle is implicit in Section 5.3 of [15][7]:

Theorem 1.7. *There is a data-structure that preprocesses $B \in \mathbb{F}[X]^{n\times n}$ where $\det(B) \neq 0$ and $\deg(B) \leq d$ in $\tilde{O}(dn^\omega)$ operations. Then, given any $\vec{v} \in \mathbb{F}[X]^n$ where $\deg(\vec{v}) \leq d$, it can compute $\vec{v}^\top \mathrm{adj}(B)$ in $\tilde{O}(dn^2)$ operations.*

However, to get $o(Wn^2)$ query time as in Theorem 1.1, Theorem 1.7 is not enough. Fortunately, by modifying the technique from [15] in a white-box manner (see Section III for details), we can obtain the following trade-off which is essential for Theorem 1.1. The result essentially interpolates the exponents of the following two extremes: $\tilde{O}(dn^3)$ preprocessing and $O(dn)$ query time when computing the adjoint explicitly, or $\tilde{O}(dn^\omega)$ preprocessing and $O(dn^2)$ query time when using Theorem 1.7.

Theorem 1.8. *For any $0 \leq \mu \leq 1$, there is a data-structure that preprocesses $B \in \mathbb{F}[X]^{n\times n}$ where $\det(B) \neq 0$ and $\deg(B) \leq d$ in $O(dn^{\omega+(3-\omega)\mu})$ operations. Then, given any pair (i, j), it returns $\mathrm{adj}(B)_{i,j}$ in $O(dn^{2-\mu})$ operations.*

To see the main idea, we give a slightly oversimplified description of the oracle in Theorem 1.7 which allows us to show how to modify the technique to obtain Theorem 1.8. In Figure 1, we write a number for each matrix entry to indicate a bound on the degree, e.g. when we write $(4, 4, 4)^\top$ then we mean a 3-dimensional vector with entries of degree at most 4. Suppose that we are now working with an $n \times n$ matrix B of degree d. Then [15] (and [14] for a special type of matrices) is able to find a full-rank matrix A of degree d in $\tilde{O}(dn^\omega)$ field operations, such that BA is a block-diagonal matrix as in Figure 1. In Figure 1 the empty sections of the matrices represent zeros in the matrix and the ds represent entries of degree at most d. This means the left part of A is a *kernel-base* of the lower part of B and likewise the right part of A is a *kernel-base* of the lower part of B.

This procedure can now be repeated on the two smaller $n/2 \times n/2$ matrices of degree $2d$. After $\log n$ such iterations we have that $B\prod_{i=1}^{\log n} A_i =: D$ is a diagonal matrix of degree dn as in Figure 2. We call this chain $A_1, \ldots, A_{\log n}$ the *kernel basis decomposition* of B. Here each A_i consists of 2^{i-1} block matrices on the diagonal of dimension $n/2^{i-1}$ and degree $d2^{i-1}$. So while the degree of these blocks doubles, the dimension is halved, which implies that all these A_i can be computed in just $\tilde{O}(dn^\omega)$ operations.

The inverse B^{-1} can be written as $\prod_{i=1}^{\log n} A_i D^{-1}$. Also, D is a diagonal matrix and so it is easily invertible, i.e. we can write the entries of the inverse in the form of rationals p/q where both p and q are of degree $O(dn)$. Therefore, we can represent the adjoint via $\mathrm{adj}(B) = B^{-1}\det(B) = \prod_i A_i D^{-1}\det(B)$.

To compute $v^\top \mathrm{adj}(B)$ for any degree d vector v in $\tilde{O}(dn^2)$ operations, we must compute $v^\top \prod_i A_i \det(B) D^{-1}$ from left to right. Each vector matrix product with some A_i has degree $d2^{i-1}$ but at the same time the dimension of the diagonal blocks is only $n/2^{i-1}$, hence each product requires only $\tilde{O}(dn^2)$ field operations. Scaling by $\det(B)$ and dividing by the entries of D also requires only $O(dn^2)$ as their degrees are bounded by $O(dn)$. This gives us Theorem 1.7.

The idea of Theorem 1.8 is to explicitly precompute a prefix $P = \prod_{i\leq k} A_i$ from the factors of $\prod_i A_i \det(B) D^{-1}$. This increases the preprocessing time but at the same time allows us to compute $\mathrm{adj}(B)_{i,j} = \vec{e}_i^\top P \prod_{i>k} A_i \det(B) D^{-1} \vec{e}_j$ faster.

B. Comparison with previous works

Previous dynamic matrix algorithms.: In contrast to algorithms for sensitive oracles that handle a single batch of updates, *dynamic* algorithms must handle an (infinite) sequence of updates. The techniques we used for our sensitive distance/reachability oracles are motivated from techniques developed for dynamic algorithms which we will discuss below.

There is a line of work initiated by Sankowski [28], [29], [30] on maintaining inverse or adjoint of a dynamic matrix whose entries are field elements, not polynomials as in our setting. Let us call such matrix a *non-polynomial matrix*. By the similar reductions for obtaining applications on weighted graphs in this paper, dynamic non-polynomial matrix algorithms imply solutions to many dynamic algorithms on *unweighted* graphs.

Despite the similarity of the results and applications, there is a sharp difference at the core techniques of our algorithm for polynomial matrices and the previous algorithms for non-polynomial matrices. The key to all our results is fast preprocessing time. By using the kernel basis decomposition [15], we do not need to explicitly write down the adjoint

[7]Section 5.3 of [15] discusses computing $v^\top A^{-1}$, where the result is given by a vector u with entries of the form p/q where p, q are polynomials of degree at most $O(dn)$. Since $\det(A)$ can be computed in $\tilde{O}(dn^\omega)$ [26], [27] and $\mathrm{adj}(A) = A^{-1}\det(A)$, we get Theorem 1.7.

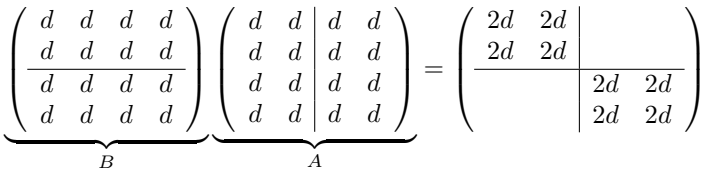

Figure 1. Nonzero entries represent the degrees. The left (right) half of A is a kernel base of the bottom (top) half of B.

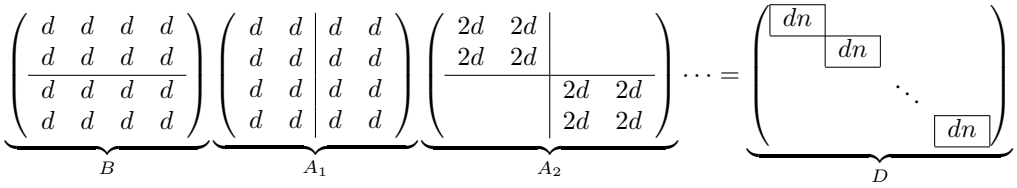

Figure 2. Kernel basis decomposition of B. Each A_i is a block-matrix consisting of kernel bases for the blocks of $B\prod_{k=1}^{i} A_k$. The product $B\prod_{i=1}^{\log n} A_i$ yields a diagonal matrix D of degree dn.

of a polynomial matrix, which takes $\Omega(dn^3)$ operations if the matrix has size $n \times n$ and degree d. This technique is specific for polynomial matrix and does not have a meaningful counterpart for a non-polynomial matrix. On the contrary, algorithms for non-polynomial matrices from [28], [29], [30] just preprocess a matrix in a trivial way. That is, they compute the inverse and/or adjoint explicitly in $O(n^\omega)$ operations. Their key contribution is how to handle update in $o(n^2)$ operations.

We remark that Sankowski [31] did obtain a dynamic polynomial matrix algorithm by extending previous dynamic non-polynomial matrix algorithms. However, there are two limitations to this approach. First, the algorithm requires a matrix of the form $(\mathbb{I} - X \cdot A)$ where $A \in \mathbb{F}[X]^{n\times n}$. This restriction excludes some applications including distances on graphs with zero or negative weights because we cannot use the reduction Lemma 2.1. Second, the cost of the algorithm is multiplied by the degree of the adjoint matrix which is $O(dn)$ if A has degree d. Hence, just to update one entry, this takes $O(dn \times n^{1.407})$ operations[8]. This is already slower than the time for computing from scratch an entry of adjoint/inverse $\tilde{O}(dn^\omega)$ using static algorithms[9].

Previous sensitive distance oracles.: Previous sensitive distance oracles such as [13], [11] also use fast matrix-multiplication, but only use it for computing a fast min-plus matrix product in a black box manner. All further techniques used by these algorithms are graph theoretic.

[8]The current best algorithm [30] takes $O(n^{1.407})$ operations to update one entry of a non-polynomial matrix.

[9]The first limitation explains why there is only one application in [31], which is to maintain distances on unweighted graphs. To bypass the second limitation, Sankowski [31] "forces" the degrees to be small by executing all arithmetic operations under modulo X^k for some small k. A lot of information about the adjoint is lost from doing this. However, for his specific application, he can still return the queried distances by combining with other graph-theoretic techniques.

Our shift from graph-theoretic techniques to a purely algebraic algorithm is the key that enables us to support large sets of updates. Let us explain why previous techniques can inherently handle only small number of deletions. Their main idea is to sample many smaller subgraphs in the preprocessing. To answer a query in the updated graph, their algorithms simply look for a subgraph H where (i) all deleted edges were not even in H from the beginning, and (ii) all edges in the new shortest path are in H. To argue that H exists with a good probability, the number of deletions cannot be more than $\log n$ where n is the number of nodes. That is, these algorithms do not really re-compute the new shortest paths, instead they pre-compute subgraphs that "avoid" the updates.

Purely algebraic algorithms such as ours (and also [31], [30], [32]) can overcome the limit on deletions naturally. For an intuitive explanation consider the following simplified example for unweighted graphs: Let A be the adjacency matrix of an unweighted graph, then the polynomial matrix $(\mathbb{I} - X \cdot A)$ has the following inverse when considering the field of formal power series: $(\mathbb{I} - X \cdot A)^{-1} = \sum_{k\geq 0} X^k A^k$ (this can be seen by multiplying both sides with $\mathbb{I} - X \cdot A$). This means the coefficient of X^k of $(\mathbb{I} - X \cdot A)^{-1}_{i,j}$ is exactly the number of walks of length k from i to j. So the entry $(\mathbb{I} - X \cdot A)^{-1}_{i,j}$ does not just tell us the distance between i and j, the entry actually encodes *all possible walks* from i to j. Thus finding a replacement path, when some edge is removed, becomes very simple because the information of the replacement path is already contained in entry $(\mathbb{I} - X \cdot A)^{-1}_{i,j}$. The only thing we are left to do is to remove all paths from $(\mathbb{I} - X \cdot A)^{-1}_{i,j}$ that use any of the removed edges. This is done via cancellations caused by applying the Sherman-Morrison formula.

Our algorithm exploits the adjoint instead of the inverse,

but the interpretation is similar since for invertible matrices the adjoint is just a scaled inverse: $\mathrm{adj}(M) = M^{-1}\det(M)$. We also do not perform the computations over $\mathbb{Z}[X]$, but $\mathbb{F}[X]$ to bound the required bit-length to represent the coefficients.

C. Organization

We first introduce relevant notations, definitions, and some known reductions in Section II. We construct the adjoint oracles from Theorems 1.7 and 1.8 based on kernel basis decomposition in Section III. Finally, we show our algorithms for maintaining adjoint of polynomial matrices in Section IV-A, where we will also apply the reductions to get our distance and reachability oracles Theorems 1.1 and 1.2.

II. Preliminaries

Complexity Measures: Most of our algorithms work over any field $\mathbb{F}$ and their complexity is measured in the number of arithmetic operations performed over $\mathbb{F}$, i.e. the *arithmetic complexity*. This does not necessarily equal the *time complexity* of the algorithm as one arithmetic operation could require more than $O(1)$ time, e.g. very large rational numbers could require many bits for their representation. This is why our algebraic lemmas and theorems will always state "in $O(\cdot)$ operations" instead of "in $O(\cdot)$ time".

For the graph applications however, when having an n node graph, we will typically use the field $\mathbb{Z}_p$ for some prime p of order n^c for some constant c. This means each field element requires only $O(\log n)$ bits to be represented and all field operations can be performed in $O(1)$ time in the standard model (or $\tilde{O}(1)$ bit-operations).

Notation: Identity and Submatrices: The identity matrix is denoted by $\mathbb{I}$.

Let $I, J \subset [n] := \{1, ..., n\}$ and A be a $n \times n$ matrix, then the term $A_{I,J}$ denotes the submatrix of A consisting of the rows I and columns J. For some $i \in [n]$ we may also just use the index i instead of $\{i\}$. The term $A_{[n],i}$ thus refers to the ith column of A.

Matrix Multiplication: We denote with $O(n^\omega)$ the arithmetic complexity of multiplying two $n \times n$ matrices. Currently the best bound is $\omega < 2.3729$ [33], [34].

Polynomial operations: Given two polynomials $p, q \in \mathbb{F}[X]$ with $\deg(p), \deg(q) \leq d$, we can add and subtract the two polynomials in $O(d)$ operations in $\mathbb{F}$. We can multiply the two polynomials in $O(d \log d)$ using fast-fourier-transformations, likewise dividing two polynomials can be done in $O(d \log d)$ as well [35, Section 8.3]. Since we typically hide polylog factors in the $\tilde{O}(\cdot)$ notation, all operations using degree d polynomials from $\mathbb{F}[X]$ can be performed in $\tilde{O}(d)$ operations in $\mathbb{F}$.

Polynomial Matrices: We will work with polynomial matrices/vectors, so matrices and vectors whose entries are polynomials. We define for $M \in \mathbb{F}[X]^{n \times m}$ the degree $\deg(M) := \max_{i,j} \deg(M_{i,j})$. Note that a polynomial matrix $M \in \mathbb{F}[X]^{n \times n}$ with $\det(A) \neq 0$ might not have an inverse in $\mathbb{F}[X]^{n \times n}$ as $\mathbb{F}[X]$ is a ring. However, the inverse M^{-1} does exist in $\mathbb{F}(X)^{n \times n}$ where $\mathbb{F}(X)$ is the field of rational functions.

Adjoint of a Matrix: The adjoint of an $n \times n$ matrix M is defined as $\mathrm{adj}(M)_{i,j} = (-1)^{i+j}\det(M_{[n]\setminus j,[n]\setminus i})$. In the case that M has non-zero determinant, we have $\mathrm{adj}(M) = \det(M) \cdot M^{-1}$. Note that in the case of M being a degree d polynomial matrix, we have $\mathrm{adj}(M) \in \mathbb{F}[X]^{n \times n}$ and $\deg(\mathrm{adj}(M)) < nd$.

Graph properties from polynomial matrices: Polynomial matrices can be used to obtain graph properties such as the distance between any pair of nodes:

Lemma 2.1 ([36, Theorem 5 and Theorem 7])**.** *Let $\mathbb{F} := \mathbb{Z}_p$ be a field of size $p \sim n^c$ for some constant $c > 1$ and let G be a graph with n nodes and integer edge weights $(c_{i,j})_{1 \leq i,j \leq n} \in [-W, W]$.*

Let $A \in \mathbb{F}[X]^{n \times n}$ be a polynomial matrix, where $A_{i,i} = X^W$ and $A_{i,j} = a_{i,j}X^{W+c_{i,j}}$ and each $a_{i,j} \in \mathbb{F}$ is chosen independently and uniformly at random.

- *If G contains no negative cycle, then the smallest degree of the non-zero monomials of $\mathrm{adj}(A)_{i,j}$ minus $W(n-1)$ is the length of the shortest path from i to j in G with probability at least $1 - n^{1-c}$.*
- *Additionally with probability at least $1 - n^{1-c}$, the graph G has a negative cycle, if and only if $\det(A)$ has a monomial of degree less than Wn.*

III. Adjoint Oracle

In this section we will outline how the adjoint oracle Theorem 1.7 by [15] can be extended to our Theorem 1.8.

Unfortunately this new result is not a blackbox reduction, instead we have to fully understand and exploit the properties of the algorithm presented in [15]. This is why a formally correct proof of Theorem 1.8 requires us to repeat many definitions and lemmas from [15]. Such a formally correct proof can be found in subsection III-B. We will start with a high level description based on the high level idea of Theorem 1.7 presented in Section I-A.

A. Extending the Oracle to Element Queries

We will now outline how the data-structure of kernel-bases, presented in Section I-A, can be used for faster element queries to $\mathrm{adj}(B)$. Remember that Theorem 1.7 was based on representing $\mathrm{adj}(B) = \prod_{i=1}^{\log n} A_i D^{-1} \det(B)$, where each A_i consists of 2^{i-1} diagonal blocks of size $n/2^{i-1} \times n/2^{i-1}$ and is of degree $d2^{i-1}$.

The idea for Theorem 1.8 is very simple: Choose some $0 \leq \mu \leq 1$ and k such that $2^k = n^\mu$, then dur-

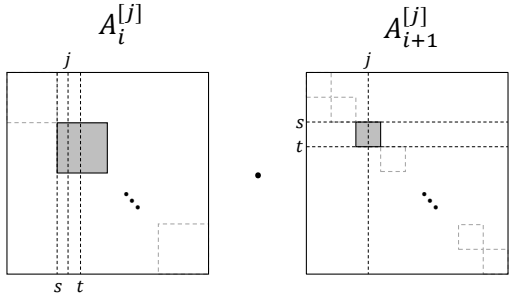

Figure 3. Dark grey boxes represent non-zero entries. The grey dotted squares represent non-zero entries of A_i that are set to zero in $A_i^{[j]}$. Here we see that only rows with index $s, ..., t$ of $A_{i+1}^{[j]}$ are non-zero, so when computing $A_i A_{i+1}^{[j]}$ only the columns of A_i with index $s, ..., t$ need to be considered. These columns are identical in A_i and $A_i^{[j]}$ so $A_i A_{i+1}^{[j]} = A_i^{[j]} A_{i+1}^{[j]}$.

ing the preprocessing compute the kernel-base decomposition $\mathrm{adj}(B) = \prod_i A_i D^{-1} \det(B)$ and pre-compute the product $M := \prod_{i=1}^{k} A_i$ explicitly. When an entry (i, j) of $\mathrm{adj}(B)$ is required, we only have to compute $\vec{e}_i^\top M \prod_{i>k} A_i \vec{e}_j D_{j,j}^{-1} \det(B)$.

Complexity of the data-structure: Let $A_i^{[j]}$ be the $(n/2^{i-1}) \times (n/2^{i-1})$ matrix obtained when setting all of A_i to 0, except for the diagonal block that includes the jth column. We will now argue, that $\vec{e}_i^\top M \prod_{r>k} A_r \vec{e}_j = \vec{e}_i^\top M \prod_{r>k} A_r^{[j]} \vec{e}_j$. This equality can be seen by computing the product from right to left:

- Consider the right-most product $A_{\log n} \vec{e}_j$. The vector $\vec{e}_j$ is non-zero only in the jth row, so only the jth column of $A_{\log n}$ matters, hence $A_{\log n} \vec{e}_j = A_{\log n}^{[j]} \vec{e}_j$.
- Consider the product $A_i A_{i+1}^{[j]}$. The matrix $A_{i+1}^{[j]}$ has few non-zero rows, so most columns of A_i will be multiplied by zero and we thus most entries of A_i do not matter for computing the product. Note that all entries of A_i that *do* matter (i.e. are multiplied with non-zero entries of $A_{i+1}^{[k]}$) are inside the block $A_i^{[j]}$, because of the recursive structure of the matrices (i.e. the blocks of A_{i+1} are obtained by splitting the blocks of A_i), see for instance Figure 3. This leads to $A_i A_{i+1}^{[j]} = A_i^{[j]} A_{i+1}^{[j]}$.

By induction we now have

$$\vec{e}_i^\top M \prod_{r>k} A_r \vec{e}_j = \vec{e}_i^\top M \prod_{r>k} A_r^{[j]} \vec{e}_j.$$

The complexity of computing this product is very low, when multiplied from left to right. Consider the first products $\vec{e}_i M A_{k+1}^{[j]}$. The degree of matrix M and $A_{k+1}^{[j]}$ are both bounded by $O(dn2^k)$. The matrix $A_{k+1}^{[j]}$ is 0 except for a $(n/2^k) \times (n/2^k)$ block on the diagonal. Hence, this first product requires $\tilde{O}(d2^k(n/2^k)^2)$ field operations.

All products after this require fewer operations: On one hand the degree of vector and matrix double after each product, on the other hand the dimension of the non-zero block of $A_r^{[j]}$ is halved. Since the complexity of the vector matrix product scales linearly in the degree but quadratic in the dimension, the complexity is bounded by the initial product $\vec{e}_i M A_{k+1}^{[j]}$. The query complexity is thus $\tilde{O}(d2^k(n/2^k)^2) = \tilde{O}(dn^{2-\mu})$.

This is only a rough simplification of how the algorithm works. For instance the degrees of the $A_1, ...A_{\log n}$ are not simple powers of 2, instead only the average degree is bounded by a power of 2. Likewise the dimension n and the size of the diagonal blocks do not have to be a power of two.

B. Formal Proof of the Adjoint Element Oracle

Before we can properly prove our Theorem 1.8, we first have to define/cite some terminology and lemmas from [15], as our theorem is heavily based on their result.

First we will define the notation of shifted column degrees. Shifted column degrees can be used to formalize how the degree of a vector changes when multiplying it with a polynomial matrix.

Definition 3.1 ([15, Section 2.2])**.** *Let $\mathbb{F}$ be some field, $M \in \mathbb{F}[X]^{n \times m}$ be some polynomial matrix and let $\vec{s} \in \mathbb{N}^n$ be some vector.*

Then the $\vec{s}$-shifted column degrees of M is defined via:

$$\mathrm{cdeg}_{\vec{s}}(M)_j := \max_{i=1,...,n} \vec{s}_i + \deg(M_{i,j}) \text{ for } j = 1, ..., m$$

Lemma 3.2. *Let $\mathbb{F}$ be some field, $M \in \mathbb{F}[X]^{n \times m}$ be some polynomial matrix and let $\vec{s} \in \mathbb{N}^n$ be some vector. Further let $\vec{v} \in \mathbb{F}[X]^n$ be a polynomial vector where $\deg(\vec{v}_i) \leq \vec{s}_i$ for $i = 1, ..., n$.*

Then $\deg((\vec{v}^\top M)_j) \leq \mathrm{cdeg}_{\vec{s}}(M)_j$ and $\vec{v}^\top M$ can be computed in $\tilde{O}(n \sum_{j=1}^{m} \mathrm{cdeg}_{\vec{s}}(M)_j)$.

Proof: We can compute the product $\vec{v}_i M_{i,j}$ using $\tilde{O}(\deg(\vec{v}_i) + \deg(M_{i,j}))$ field operations. Hence the total cost becomes

$$\begin{aligned}
&\sum_{j=1}^{m} \sum_{i=1}^{n} \tilde{O}\left(\deg(\vec{v}_i) + \deg(M_{i,j})\right) \\
\leq &\tilde{O}\left(\sum_{j=1}^{m} \sum_{i=1}^{n} \vec{s}_i + \deg(M_{i,j})\right) \\
\leq &\tilde{O}\left(n \sum_{j=1}^{m} \mathrm{cdeg}_{\vec{s}}(M)_j\right).
\end{aligned}$$

■

We will now give a formal description of the data-structure constructed in [15]. The following definitions and properties hold throughout this entire section.

Let $B \in \mathbb{F}[X]^{n\times n}, \vec{s} \in \mathbb{N}^n$ such that $\operatorname{cdeg}_{\vec{0}}(B)_j \leq \vec{s}_j$, so $\vec{s}_j$ bounds the maximum degree in the jth column of M (also called the column degree of M). Let $d := \sum_i \vec{s}_i/n$ be the average column degree of B.

In [15] they construct in $\tilde{O}(dn^\omega)$ field operations a chain of matrices $A_1, ..., A_{\lceil \log n \rceil} \in \mathbb{F}[X]^{n\times n}$ and a diagonal matrix $D \in \mathbb{F}[X]^{n\times n}$ such that $\operatorname{adj}(B) = (\prod_i A_i)\det(B)D^{-1}$.

Here the matrices $(A_{i+1})_{i=0...\lceil \log n\rceil -1}$ are block matrices consisting each of 2^i diagonal blocks, i.e.

$$A_{i+1} = \operatorname{diag}(A_{i+1}^{(1)}, ..., A_{i+1}^{(2^i)}).$$

The number of rows/column of each $A_{i+1}^{(k)}$ is $n/2^i$ upto a factor of 2. (Note that $A_i^{(k)}$ refers to the kth block on the diagonal of A_i, not to be confused with our earlier definition of $A_i^{[j]}$ in subsection III-A).

Remember from the overview (Section I-A) that each $A_i^{(k)}$ consists of two kernel bases, so each of these diagonal block matrices consists in turn of two matrices (kernel bases)

$$A_{i+1}^{(k)} = [N_{i+1,l}^{(k)}, N_{i+1,r}^{(k)}].$$

Here l and r are not variables but denote the *left* and *right* submatrix.

We also write M_i for the partial product $M_i := \prod_{k=1}^{i} A_k$. Each M_i can be decomposed into

$$M_i = [M_i^{(1)}, ..., M_i^{(2^i)}]$$

where each $M_i^{(k)}$ has n rows and the number of columns in $M_i^{(k)}$ corresponds to the number of columns in $A_{i+1}^{(k)}$. We can compute M_i as follows:

$$M_{i+1}^{(2k-1)} = M_i^{(k)} N_{i+1,l}^{(k)} \text{ and } M_{i+1}^{(2k)} = M_i^{(k)} N_{i+1,r}^{(k)} \quad (1)$$

We have the following properties for the degrees of these matrices:

Lemma 3.3 (Lemma 10 in [15])**.** *Let $m \times m$ be the dimension of $M_i^{(k)}$, let m_l and m_r be the number of columns in $N_{i+1,l}^{(k)}$ and $N_{i+1,r}^{(k)}$ respectively and let $\vec{t} := \operatorname{cdeg}_{\vec{s}}(M_i^{(k)})$, then*

- $\sum_{j=1}^{m} \vec{t}_j \leq \sum_{j=1}^{n} \vec{s}_j = dn$, *and*
- $\sum_{j=1}^{m_l} \operatorname{cdeg}_{\vec{t}}(N_{i+1,l}^{(k)})_j \leq \sum_{j=1}^{n} \vec{s}_j = dn$ *and* $\sum_{j=1}^{m_r} \operatorname{cdeg}_{\vec{t}}(N_{i+1,r}^{(k)})_j \leq \sum_{j=1}^{n} \vec{s}_j = dn$ *(which also implies* $\sum_{j=1}^{m} \operatorname{cdeg}_{\vec{t}}(A_{i+1}^{(k)})_j \leq 2dn$*)*

Lemma 3.4 (Lemma 11 in [15])**.** *For any i and k computing the matrix products in* (1) *requires $\tilde{O}(n(n/2^i)^{\omega-1}(1+d2^i)))$ field operations.*

We now have defined all the required lemmas and notation from [15] and we can now start proving Theorem 1.8.

The following lemma is analogous to Lemma 3.4, though now we want to compute only one row of product (1). This lemma will bound the complexity of the query operation in Theorem 1.8.

Lemma 3.5. *Let $v^\top := \vec{e}_r^\top M_i^{(k)}$ be some row of $M_i^{(k)}$, then we can compute $v^\top N_{i+1,c}^{(k)}$ for both $c = l, r$ in $\tilde{O}(dn^2/2^i)$.*

Proof: The matrices $N_{i+1,l}^{(k)}$ and $N_{i+1,r}^{(k)}$ form the matrix $A_{i+1}^{(k)}$, so we instead just compute the product $\vec{v}^\top A_{i+1}^{(k)}$. The matrix $A_{i+1}^{(k)}$ is of size $(n/2^i) \times (n/2^i)$ (up to a factor of 2). Let $\vec{t} := \operatorname{cdeg}(\vec{v}^\top)$ then by Lemma 3.3 we know $\sum_j \operatorname{cdeg}_{\vec{t}}(A_{i+1}^{(k)})_j \leq 2\sum_j \vec{s}_j = 2dn$ and $\sum_j \vec{t}_j \leq dn$.

By Lemma 3.2 the cost of computing the product $\vec{v}^\top A_{i+1}^{(k)}$ is $\tilde{O}(n/2^i \sum_{j=1}^{m} \operatorname{cdeg}_{\vec{t}}(A_{i+1}^{(k)})_j)$, which, given the degree bounds, can be simplified to $\tilde{O}(dn^2/2^i)$. ∎

The following lemma will bound the complexity for the preprocessing of Theorem 1.8.

Lemma 3.6. *If we already know the matrix A_i for $i = 1, ...\lceil \log n \rceil$, then for any $0 \leq \mu \leq 1$ we can compute $M_{\lceil \log n^\mu \rceil}$ in $\tilde{O}(dn^{\omega(1-\mu)+3\mu})$ field operations.*

Proof: To compute M_{i+1}, requires to compute all 2^i many $M_{i+1}^{(k)}$ for $k = 1...2^i$. Assume we already computed matrix M_i, then we can compute $M_{i+1}^{(k)}$ for $k = 1...2^i$ via Lemma 3.4. Inductively the total cost we obtain is:

$$\sum_{i=1}^{\lceil \log n^\mu \rceil} \tilde{O}(n(n/2^i)^{\omega-1}(1+d2^i)) \cdot 2^i = \tilde{O}(dn^{\omega(1-\mu)+3\mu})$$

∎

The last lemma we require for the proof of Theorem 1.8 is that we can compute the determinant of B in $\tilde{O}(dn^\omega)$ field operations.

Lemma 3.7 ([26], [27])**.** *Let $B \in \mathbb{F}[X]^{n\times n}$ be a matrix of degree at most d, then we can compute $\det(B)$ in $\tilde{O}(dn^\omega)$ field operations.*

Proof of Theorem 1.8: The claim is that for any $0 \leq \mu \leq 1$ we can, after $\tilde{O}(dn^{\omega(1-\mu)+3\mu})$ preprocessing of B, compute any entry $\operatorname{adj}(B)_{i,j}$ in $\tilde{O}(n^{2-\mu})$ operations.

Preprocessing: We first compute the determinant $\det(B)$ via Lemma 3.7 and construct the chain of matrices $A_1, ...A_{\lceil \log n \rceil}$ as in [15] in $\tilde{O}(dn^\omega)$ operations, then we compute $M_{\lceil \log n^\mu \rceil}$ in $\tilde{O}(dn^{\omega(1-\mu)+3\mu})$ operations using Lemma 3.6.

Queries: To answer a query for $\operatorname{adj}(A)_{i,j}$, we compute one entry $(M_{\lceil \log n \rceil})_{i,j}$, multiply it with $\det(A)$ and divide it by $D_{j,j}$, because $\operatorname{adj}(A) = (\prod_{i=1}^{\lceil \log n \rceil} A_i)\det(A)D^{-1} = M_{\lceil \log n \rceil}\det(A)D^{-1}$.

Here the expensive part is to compute the entry of $M_{\lceil \log n \rceil}$, which is done by computing one row of $M_{\lceil \log n \rceil}^{(k)}$ for some appropriate k. Via (1), we know

$$M_{\lceil \log n \rceil}^{(k)} = M_{\lceil \log n^\mu \rceil} \prod_{t=1}^{\lceil \log n \rceil - \lceil \log n^\mu \rceil} N_{\lceil \log n^\mu \rceil + t, c_t}^{(k_t)}$$

for some sequence $k_t \in [2^{\lceil \log n^\mu \rceil + t}]$, $c_t \in \{l, r\}$.

So we only have to compute the product of the ith row of $M_{\lceil \log n^\mu \rceil}$ with a sequence of $(N^{(k_t)}_{\lceil \log n^\mu \rceil + t, c_t})_{t=1,\dots,\lceil \log n \rceil - 1}$ matrices. Computing this product from left to right means we compute the following intermediate results

$$\vec{e}_i^\top M_{\lceil \log n^\mu \rceil} \prod_{t=1}^{r} N^{(k_t)}_{\lceil \log n^\mu \rceil + t, c_t} = \vec{e}_i^\top M^{(k_r)}_{\lceil \log n^\mu \rceil + r}$$

for every $r = 1, ..., \lceil \log n \rceil - \lceil \log n^\mu \rceil$. So each intermediate result is just the ith row of some matrix $M^{(k_r)}_{\lceil \log n^\mu \rceil + r}$. This means such a vector-matrix product can be computed via Lemma 3.5 in

$$\tilde{O}(dn^2 / 2^{\lceil \log n^\mu \rceil + r})$$

Here the first product for $r = 1$ is the most expensive and the total cost for all $\lceil \log n \rceil - \lceil \log n^\mu \rceil$ many vector-matrix products becomes $\tilde{O}(dn^{2-\mu})$.

■

IV. Sensitive Distance and Reachability Oracles

In this section we will use the adjoint oracle from Section III to obtain the results presented in Section I. A high-level description of that algorithm was already outlined in Section I-A.

This section is split into two parts: First we will describe in Section IV-A our results for maintaining the adjoint of a polynomial matrix, which will conclude with the proof of Theorem 1.1. The second subsection IV-B will explain how to obtain the sensitive reachability oracle Theorem 1.2. These graph theoretic results will be stated more accurately than in the overview by adding trade-off parameters and memory requirements.

All proofs in this section assume that the matrix A stays non-singular throughout all updates, which is the case w.h.p for matrices constructed via the reduction of Lemma 2.1.

A. Adjoint oracles

Theorem 4.1. *Let $\mathbb{F}$ be some field and $A \in \mathbb{F}[X]^{n \times n}$ be a polynomial matrix with $\det(A) \neq 0$ of degree d. Then for any $0 \leq \mu \leq 1$ we can create a data-structure storing $O(dn^{2+\mu})$ field elements in $\tilde{O}(dn^{\omega(1-\mu)+3\mu})$ field operations, such that:*

We can change f columns of A and update our data-structure in $\tilde{O}(dn^{2-\mu}f^2 + dnf^\omega)$ field operations storing additional $O(dnf^2)$ field elements. This updated data-structure allows for querying entries of $\operatorname{adj}(A)$ in $\tilde{O}(dnf^2 + dn^{2-\mu}f)$ field operations and queries to the determinant of the new A in $O(dn)$ operations.

Proof: We start with the high-level idea: We will express the change of f entries of A via the rank f update $A + UV^\top$, where $U, V \in \mathbb{F}^{n \times f}$ and both matrices have only one non-zero entry per column. Via Lemma 1.6 we know that for $M := \mathbb{I} \cdot \det(A) + V^\top \operatorname{adj}(A) U$ the new adjoint is given by

$$\operatorname{adj}(A + UV^\top) = \big(\operatorname{adj}(A) \det(M) - (\operatorname{adj}(A)U) \operatorname{adj}(M) \left(V^\top \operatorname{adj}(A)\right)\big) \det(A)^{-f}.$$

Our algorithm is as follows:

Initialization: During the initialization use Theorem 1.8 on matrix A in $\tilde{O}(dn^{\omega+(3-\omega)\mu})$ operations, which will allow us later to query entries $\operatorname{adj}(A)_{i,j}$ in $\tilde{O}(dn^{2-\mu})$ operations. We also compute $\det(A)$ via Lemma 3.7 in $\tilde{O}(dn^\omega)$ operations.

Updating the data-structure: The first task is to compute the matrix M. Note that for element updates to A, the matrices U and V have only one non-zero entry per column, so $V^\top \operatorname{adj}(A) U$ is just an $f \times f$ submatrix of A where each entry is multiplied by one non-zero entry of U and V. Thus we can compute $M = \mathbb{I} \cdot \det(A) + V^\top \operatorname{adj}(A) U$ in $\tilde{O}(df^2 n^{2-\mu})$ operations, thanks to the preprocessing of A. Next, we preprocessing the matrix M using Theorem 1.7, which requires $\tilde{O}(dnf^\omega)$ as the degree of M is bounded by $O(dn)$. We also compute $\det(M)$ in $\tilde{O}(dnf^\omega)$ operations.

Thus the update complexity is bounded by $\tilde{O}(df^2 n^{2-\mu} + dnf^\omega)$ operations.

Querying entries: To query an entry (i, j) of $\operatorname{adj}(A + UV^\top)$ we must compute

$$(\operatorname{adj}(A)_{i,j} \det(M) - (\vec{e}_i \operatorname{adj}(A) U) \cdot \operatorname{adj}(M) \left(V^\top \operatorname{adj}(A) \vec{e}_j\right)) \det(A)^{-f}.$$

Here $\operatorname{adj}(A)_{i,j} \det(M)$ can be computed in $\tilde{O}(dn^{2-\mu} + dnf)$ operations. The vectors $\vec{e}_i \operatorname{adj}(A) U$ and $V^\top \operatorname{adj}(A) \vec{e}_j$ are just f entries of $\operatorname{adj}(A)$ where each entry is multiplied by one non-zero entry of U or V, so we can compute them in $\tilde{O}(dn^{2-\mu}f)$ operations. Multiplying one of these vectors with $\operatorname{adj}(M)$ requires $\tilde{O}(dnf^2)$ operations because of the preprocessing of M via Theorem 1.7. The product of $(\vec{e}_i \operatorname{adj}(A) U) \operatorname{adj}(M)$ with $\left(V^\top \operatorname{adj}(A) \vec{e}_j\right)$ can be computed in $\tilde{O}(dnf^2)$ operations. Subtracting $\operatorname{adj}(A)_{i,j} \det(M)$ from the product and dividing by $\det(A)^f$ can be done in $\tilde{O}(dnf)$ operations. The query complexity is thus $\tilde{O}(dnf^2 + dn^{2-\mu}f)$ operations.

Maintaining the determinant: The determinant is given by $\det(A + UV^\top) = \det(M) / \det(A)^{f-1}$ (as can be seen in the proof of Section VI). The division requires only $\tilde{O}(dnf)$ operations.

■

In this section we will state the result from Section I in a more formal way. We will state trade-off parameters, memory consumption and we will separate the sensitive setting into two phases: update and query.

Theorem 4.2 (Corresponds to Theorem 1.1)**.** *Let G be a graph with n nodes and edge weights in $[-W, W]$. For any $0 \leq \mu \leq 1$ we can create in $\tilde{O}(Wn^{\omega(1-\mu)+3\mu})$ time a Monte*

Carlo data-structure that requires $O(Wn^{2+\mu}\log n)$ bits of memory, such that:

We can then change f edges (additions and removals) and update our data-structure in $\tilde{O}(Wn^{2-\mu}f^2 + Wnf^\omega)$ time using additional $O(Wnf^2\log n)$ bits of memory. This updated data-structure allows for querying the distance between any pair of nodes in $\tilde{O}(Wnf^2 + Wn^{2-\mu}f)$ time, or report that there exists a negative cycle in $O(1)$ time.

Proof of Theorem 4.2: We construct the matrix as specified in Lemma 2.1. Since the field size is polynomial in n, the arithmetic operations can be executed in $O(1)$ time in the standard model and saving one field element requires $O(\log n)$ bits.

The determinant of the matrix is computed during the update, where we will also check for negative cycles by checking if the determinant has a non-zero monomial of degree less than Wn. This way we can answer the existence of a negative cycle in $O(1)$ time, when a query to a distance is performed. Theorem 4.2 is now implied by Theorem 4.1. ∎

B. Sensitive reachability oracle

So far we have only proven the distance applications. Using the same techniques the result can also be extended to a sensitive reachability oracle.

Reachability can be formulated as a distance problem where every edge has cost 0. In that case the matrix constructed in Lemma 2.1 has degree 0, so the matrix is in $\mathbb{F}^{n\times n}$ and we do not have to bother with polynomials anymore. The algorithm for the following result is essentially the same as Theorem 4.1, but since we no longer have polynomial matrices, we no longer require more sophisticated adjoint oracles and can instead use simpler subroutines.

Theorem 4.3 (Corresponds to Theorem 1.2)**.** *Let G be a graph with n nodes. We can construct in $O(n^\omega)$ time a Monte Carlo data-structure that requires $O(n^2\log n)$ bits of memory, such that:*

We can change any f edges (additions and removals) and update our data-structure in $O(f^\omega)$ time using additional $O(f^2\log n)$ bits of memory. This updated data-structure allows for querying the reachability between any pair of nodes in $O(f^2)$ time.

Proof: We will now construct a data-structure that, after some initial preprocessing of some matrix $A \in \mathbb{F}^{n\times n}$, allows us to quickly query elements of $\operatorname{adj}(A + UV^\top)$ for any sparse $U, V \in \mathbb{F}^{n\times f}$ where U and V have at most one non-zero entry per column.

Preprocessing: We compute $\det(A)$ and $\operatorname{adj}(A)$ explicitly in $O(n^\omega)$ field operations.

Update: We receive matrices $U, V \in \mathbb{F}^{f\times n}$, where each column has only one non-zero entry. We compute $V^\top \operatorname{adj}(A)U$ in $O(f^2)$ operations because of the sparsity of U and V. This means we can now compute for $M := \mathbb{I}\cdot\det(A) + V^\top \operatorname{adj}(A)U$ the matrix $\operatorname{adj}(M)$ and $\det(M)$ in $O(f^\omega)$ operations.

Query: When querying an entry (i,j) of $\operatorname{adj}(A+UV^\top)$ we have to compute:

$$\frac{\operatorname{adj}(A)_{i,j}\det(M) - \operatorname{adj}(A)_{i,[n]}U\ \operatorname{adj}(M)\ V^\top \operatorname{adj}(A)_{[n],j}}{\det(A)^f}$$

The vectors $\vec{u}^\top := \operatorname{adj}(A)_{i,[n]}U$ and $\vec{v} := V^\top \operatorname{adj}(A)_{[n],j}$ can be computed in $O(f)$ operations as they are just $O(f)$ entries of the adjoint, each multiplied by one non-zero entry of U or V. The product $\vec{u}^\top \operatorname{adj}(M)\vec{v}$ can be computed in $O(f^2)$ operations. Thus the entry of $\operatorname{adj}(A+UV^\top)$ can be computed in $O(f^2)$ operations. ∎

V. Open Problems

We present the first graph application of the kernel basis decomposition by [14], [15], and we are interested in seeing if there are further uses of that technique outside of the symbolic computation area.

Our sensitive distance oracle has subqubic preprocessing time and subquadratic query time. When restricting to only a single edge deletion, Grandoni and V. Williams were able to improve this to subqubic preprocessing and sublinear query time [11]. An interesting open question is whether a similar result can be obtained for multiple edge deletions. Even supporting just two edge deletions with subqubic preprocessing and sublinear query time would a good first step. Alternatively, disproving the existence of such a data-structure would also be interesting. Currently the best (conditional) lower bound by V. Williams and Williams [37], [24] refutes such an algorithm, if its complexity has a polylog(W) dependency on the largest edge weight W. For algorithms with poly(W) dependency no such lower bound is known.

All our data-structures maintain only the distance, or in case of reachability, return a boolean answer. So another open problem would be to find a data-structure, that does not just return the distance, but also the actual path.

Finally, our data-structures are randomized Monte-Carlo. We wonder if there is a deterministic equivalent. In case of the previous result by Weimann and Yuster [13], a deterministic equivalent was found by Alon et al. [38].

VI. Proof of Lemma 1.6

Proof of Lemma 1.6: Via the Sherman-Morrison-Woodbury formula we have

$$(A+UV^\top)^{-1} = A^{-1} - A^{-1}U(\mathbb{I}+V^\top A^{-1}U)^{-1}V^\top A^{-1}$$

and via the Sylvester determinant identity $\det(\mathbb{I}+AB) = \det(\mathbb{I}+BA)$ we have

$$\begin{aligned}\det(A+UV^\top) &= \det(A)\cdot\det(\mathbb{I}+A^{-1}UV^\top)\\ &= \det(A)\cdot\det(\mathbb{I}+V^\top A^{-1}U).\end{aligned}$$

This allows us to write the determinant of $M := \mathbb{I} \cdot \det(A) + V^\top \operatorname{adj}(A) U$ as follows:

$$\begin{aligned}\det(M) &= \det(\mathbb{I} \cdot \det(A) + V^\top \operatorname{adj}(A) U) \\ &= \det(A)^f \cdot \det(\mathbb{I} + V^\top A^{-1} U) \\ &= \det(A + UV^\top) \cdot \det(A)^{f-1}\end{aligned}$$

Which yields $\operatorname{adj}(A) \cdot \det(M) \det(A)^{-f} = A^{-1} \det(A + UV^\top)$ because $\operatorname{adj}(A) = A^{-1} \det(A)$. Likewise we obtain:

$$\begin{aligned}&\frac{(\operatorname{adj}(A)U)\ \operatorname{adj}(M)\ \left(V^\top \operatorname{adj}(A)\right)}{\det(A)^f} \\ =&\frac{\begin{array}{r}\det(M) \det(A)^2 \left(A^{-1}U\right) \\ \cdot (\mathbb{I} \cdot \det(A) + V^\top \operatorname{adj}(A) U)^{-1}\ \left(V^\top A^{-1}\right)\end{array}}{\det(A)^f} \\ =&\frac{\det(M) \det(A) \left(A^{-1}U\right)\ (\mathbb{I} + V^\top A^{-1} U)^{-1}\ \left(V^\top A^{-1}\right)}{\det(A)^f} \\ =&\det(A + UV^\top) \left(A^{-1}U\right)\ (\mathbb{I} + V^\top A^{-1} U)^{-1}\ \left(V^\top A^{-1}\right)\end{aligned}$$

Thus we obtain Lemma 1.6 by multiplying the Sherman-Morrison-Woodbury identity with $\det(A + UV^\top)$. ∎

ACKNOWLEDGMENT

This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme under grant agreement No 715672. This work was mostly done while Thatchaphol Saranurak was at KTH Royal Institute of Technology.

REFERENCES

[1] M. Patrascu and M. Thorup, "Planning for fast connectivity updates," in *FOCS*. IEEE Computer Society, 2007, pp. 263–271.

[2] R. Duan and S. Pettie, "Connectivity oracles for failure prone graphs," in *STOC*. ACM, 2010, pp. 465–474.

[3] ——, "Connectivity oracles for graphs subject to vertex failures," in *SODA*. SIAM, 2017, pp. 490–509.

[4] M. Henzinger and S. Neumann, "Incremental and fully dynamic subgraph connectivity for emergency planning," in *ESA*, ser. LIPIcs, vol. 57. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 48:1–48:11.

[5] S. Baswana and N. Khanna, "Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs," *Algorithmica*, vol. 66, no. 1, pp. 18–50, 2013.

[6] S. Chechik, M. Langberg, D. Peleg, and L. Roditty, "f-sensitivity distance oracles and routing schemes," *Algorithmica*, vol. 63, no. 4, pp. 861–882, 2012, announced at ESA'10.

[7] S. Chechik, S. Cohen, A. Fiat, and H. Kaplan, "$(1+\epsilon)$-approximate f-sensitive distance oracles," in *SODA*. SIAM, 2017, pp. 1479–1496.

[8] C. Demetrescu, M. Thorup, R. A. Chowdhury, and V. Ramachandran, "Oracles for distances avoiding a failed node or link," *SIAM J. Comput.*, vol. 37, no. 5, pp. 1299–1318, 2008, announced at SODA'02 and ISAAC'02.

[9] A. Bernstein and D. R. Karger, "Improved distance sensitivity oracles via random sampling," in *SODA*. SIAM, 2008, pp. 34–43.

[10] ——, "A nearly optimal oracle for avoiding failed vertices and edges," in *STOC*. ACM, 2009, pp. 101–110.

[11] F. Grandoni and V. V. Williams, "Improved distance sensitivity oracles via fast single-source replacement paths," in *FOCS*. IEEE Computer Society, 2012, pp. 748–757.

[12] R. Duan and S. Pettie, "Dual-failure distance and connectivity oracles," in *SODA*. SIAM, 2009, pp. 506–515.

[13] O. Weimann and R. Yuster, "Replacement paths and distance sensitivity oracles via fast matrix multiplication," *ACM Trans. Algorithms*, vol. 9, no. 2, pp. 14:1–14:13, 2013.

[14] C. Jeannerod and G. Villard, "Essentially optimal computation of the inverse of generic polynomial matrices," *J. Complexity*, vol. 21, no. 1, pp. 72–86, 2005.

[15] W. Zhou, G. Labahn, and A. Storjohann, "A deterministic algorithm for inverting a polynomial matrix," *J. Complexity*, vol. 31, no. 2, pp. 162–173, 2015.

[16] T. Lengauer and R. E. Tarjan, "A fast algorithm for finding dominators in a flowgraph," *ACM Trans. Program. Lang. Syst.*, vol. 1, no. 1, pp. 121–141, 1979.

[17] A. L. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, R. E. Tarjan, and J. Westbrook, "Linear-time algorithms for dominators and other path-evaluation problems," *SIAM J. Comput.*, vol. 38, no. 4, pp. 1533–1573, 2008.

[18] L. Georgiadis and R. E. Tarjan, "Dominators, directed bipolar orders, and independent spanning trees," in *ICALP (1)*, ser. Lecture Notes in Computer Science, vol. 7391. Springer, 2012, pp. 375–386.

[19] W. Fraczak, L. Georgiadis, A. Miller, and R. E. Tarjan, "Corrections to "finding dominators via disjoint set union" [J. discrete algorithms 23 (2013) 2-20]," *J. Discrete Algorithms*, vol. 26, pp. 106–110, 2014.

[20] K. Choudhary, "An optimal dual fault tolerant reachability oracle," in *ICALP*, ser. LIPIcs, vol. 55. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 130:1–130:13.

[21] S. Baswana, K. Choudhary, and L. Roditty, "Fault tolerant reachability for directed graphs," in *DISC*, ser. Lecture Notes in Computer Science, vol. 9363. Springer, 2015, pp. 528–543.

[22] ——, "Fault tolerant subgraph for single source reachability: generic and optimal," in *STOC*. ACM, 2016, pp. 509–518.

[23] A. Abboud and V. V. Williams, "Popular conjectures imply strong lower bounds for dynamic problems," in *FOCS*. IEEE Computer Society, 2014, pp. 434–443.

[24] M. Henzinger, A. Lincoln, S. Neumann, and V. V. Williams, "Conditional hardness for sensitivity problems," in *ITCS*, ser. LIPIcs, vol. 67. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, pp. 26:1–26:31.

[25] P. Bürgisser, M. Clausen, and M. A. Shokrollahi, *Algebraic complexity theory*, ser. Grundlehren der mathematischen Wissenschaften. Springer, 1997, vol. 315.

[26] A. Storjohann, "High-order lifting and integrality certification," *J. Symb. Comput.*, vol. 36, no. 3-4, pp. 613–648, 2003.

[27] G. Labahn, V. Neiger, and W. Zhou, "Fast, deterministic computation of the hermite normal form and determinant of a polynomial matrix," *J. Complexity*, vol. 42, pp. 44–71, 2017.

[28] P. Sankowski, "Dynamic transitive closure via dynamic matrix inverse (extended abstract)," in *FOCS*. IEEE Computer Society, 2004, pp. 509–517.

[29] ——, "Faster dynamic matchings and vertex connectivity," in *SODA*. SIAM, 2007, pp. 118–126.

[30] J. van den Brand, D. Nanongkai, and T. Saranurak, "Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds," *CoRR*, vol. abs/1905.05067, 2019, to be announced at FOCS'19.

[31] P. Sankowski, "Subquadratic algorithm for dynamic shortest distances," in *COCOON*, ser. Lecture Notes in Computer Science, vol. 3595. Springer, 2005, pp. 461–470.

[32] J. van den Brand and D. Nanongkai, "Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time," *Manuscript*, 2019, to be announced at FOCS'19.

[33] F. L. Gall, "Powers of tensors and fast matrix multiplication," in *ISSAC*. ACM, 2014, pp. 296–303.

[34] V. V. Williams, "Multiplying matrices faster than coppersmith-winograd," in *STOC*. ACM, 2012, pp. 887–898.

[35] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[36] P. Sankowski, "Shortest paths in matrix multiplication time," in *ESA*, ser. Lecture Notes in Computer Science, vol. 3669. Springer, 2005, pp. 770–778.

[37] V. V. Williams and R. R. Williams, "Subcubic equivalences between path, matrix, and triangle problems," *J. ACM*, vol. 65, no. 5, pp. 27:1–27:38, 2018, announced at FOCS'10.

[38] N. Alon, S. Chechik, and S. Cohen, "Deterministic combinatorial replacement paths and distance sensitivity oracles," in *ICALP*, ser. LIPIcs, vol. 132. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019, pp. 12:1–12:14.