Parameterized Sensitivity Oracles and Dynamic Algorithms Using Exterior Algebras

Josh Alman
Department of Computer Science, Columbia University, New York, NY, USA

Dean Hirsch
Department of Computer Science, Columbia University, New York, NY, USA

Abstract

We design the first efficient sensitivity oracles and dynamic algorithms for a variety of parameterized problems. Our main approach is to modify the algebraic coding technique from static parameterized algorithm design, which had not previously been used in a dynamic context. We particularly build off of the "extensor coding" method of Brand, Dell and Husfeldt [STOC'18], employing properties of the exterior algebra over different fields.

For the k-Path detection problem for directed graphs, it is known that no efficient dynamic algorithm exists (under popular assumptions from fine-grained complexity). We circumvent this by designing an efficient sensitivity oracle, which preprocesses a directed graph on n vertices in $2^k \operatorname{poly}(k) n^{\omega+o(1)}$ time, such that, given ℓ updates (mixing edge insertions and deletions, and vertex deletions) to that input graph, it can decide in time $\ell^2 2^k \operatorname{poly}(k)$ and with high probability, whether the updated graph contains a path of length k. We also give a deterministic sensitivity oracle requiring $4^k \operatorname{poly}(k) n^{\omega+o(1)}$ preprocessing time and $\ell^2 2^{\omega k + o(k)}$ query time, and obtain a randomized sensitivity oracle for the task of approximately counting the number of k-paths. For k-Path detection in undirected graphs, we obtain a randomized sensitivity oracle with $O(1.66^k n^3)$ preprocessing time and $O(\ell^3 1.66^k)$ query time, and a better bound for undirected bipartite graphs.

In addition, we present the first fully dynamic algorithms for a variety of problems: k-Partial Cover, m-Set k-Packing, t-Dominating Set, d-Dimensional k-Matching, and Exact k-Partial Cover. For example, for k-Partial Cover we show a randomized dynamic algorithm with $2^k \operatorname{poly}(k) \operatorname{polylog}(n)$ update time, and a deterministic dynamic algorithm with $4^k \operatorname{poly}(k) \operatorname{polylog}(n)$ update time. Finally, we show how our techniques can be adapted to deal with natural variants on these problems where additional constraints are imposed on the solutions.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Fixed parameter tractability; Mathematics of computing → Paths and connectivity problems

Keywords and phrases sensitivity oracles, k-path, dynamic algorithms, parameterized algorithms, set packing, partial cover, exterior algebra, extensor, algebraic algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.9

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2204.10819

Funding This research was supported in part by a grant from the Simons Foundation (Grant Number 825870 JA), by NSF grant CCF-2107187, by a grant from the Columbia-IBM center for Blockchain and Data Transparency, by JPMorgan Chase & Co., and by LexisNexis Risk Solutions. Any views or opinions expressed herein are solely those of the authors listed.

1 Introduction

The area of dynamic algorithms studies how to quickly and efficiently solve computational problems when the input data is changing. For example, if P is a property of a graph, then a dynamic graph algorithm for P is a data structure which maintains an n-node graph G, and can handle updates which insert or remove an edge of G, and queries which ask whether G currently satisfies P. Efficient dynamic algorithms, which handle updates and queries in $n^{o(1)}$ time, have been designed for many important problems, and they are used in many applications, both as ways to analyze evolving data, and as subroutines of larger algorithms which need to iterate over and check many similar possibilities. See, for instance, the recent survey [16].

However, there are many prominent dynamic problems which would have many applications, but for which we do not have efficient algorithms. Often times, we even have conditional lower bounds from fine-grained complexity, showing that efficient dynamic algorithms for these problems are unlikely to exist (see e.g. [1, 17, 23] and [16, Section 2.1]). It has recently become popular to circumvent such lower bounds by instead designing a *sensitivity oracle* for the problem, a weaker notion which can still be used in many applications.

Let ℓ be a positive integer. A sensitivity oracle for a dynamic problem, with sensitivity ℓ, preprocesses an initial input, and must answer queries where $\leq \ell$ changes are made to the *initial* input. For example, if P is a property of a graph, then a graph algorithm for P with sensitivity ℓ is a data structure which preprocesses an initial graph G, and can handle queries where ℓ edges are updated (inserted or removed) in the initial graph G, and asks whether P is still satisfied. One can imagine "resetting" G back to its original state after each query[1].

In this paper, we study dynamic algorithms and sensitivity oracles for *parameterized* problems. Consider, for instance, the k-PATH problem: given a positive integer k, in an n-node graph G (directed or undirected), determine whether there is a path of length k. This problem is NP-complete, so we should not hope for a dynamic algorithm with update time $n^{o(1)}$ (such a dynamic algorithm could be used to solve the static problem in $n^{2+o(1)}$ time!). However, k-PATH is known to be fixed-parameter tractable (FPT), and can be solved in time $2^{O(k)} \cdot n^2$ [8, 26, 25], which is sufficiently efficient when k is small. We can thus hope for dynamic parameterized algorithms for the problem, with update time $f(k) \cdot n^{o(1)}$. And indeed, a recent line of work has designed efficient dynamic parameterized algorithms with such a running time for many different problems, typically by using dynamic variants on classic techniques from the parameterized algorithms literature like kernelization and color coding. For the k-PATH problem in *undirected* graphs, such an algorithm is known with update time $k! \cdot 2^{O(k)} \cdot \text{polylog}(n)$ [2], and another with amortized update time $2^{O(k^2)}$ [11].

By contrast, no efficient dynamic parameterized algorithm for k-PATH in *directed* graphs is known. Moreover, Alman, Mnich and Vassilevska [2] proved a conditional lower bound, that it does not have such an efficient dynamic parameterized algorithm assuming any one of three popular conjectures from fine-grained complexity theory (the 3SUM conjecture, the TRIANGLE conjecture, and a "LAYERED REACHABILITY ORACLE" conjecture they introduce, which concerns a special case of a more popular hypothesis about reachability oracles).

This leads naturally to the two main questions we address in this paper. The first asks whether there is an analogue of the aforementioned line of work on sensitivity oracles for problems without efficient dynamic algorithms in the parameterized setting.

▶ **Question 1.** *Is there an efficient parameterized sensitivity oracle for k-PATH in directed graphs?*

[1] Sensitivity oracles are sometimes referred to as "fault-tolerant" or "emergency planning" algorithms in the literature. For graph problems, these terms also sometimes refer to the decrement-only case (where edge updates only remove edges), but following [18, Section A.1], we use "sensitivity oracle" to refer to the fully dynamic case, where edges can be inserted and deleted.

We say that a sensitivity oracle for a parameterized problem, with parameter k, is *efficient* if its preprocessing time is $f(k) \cdot \text{poly}(n)$ for some computable function f, and its query time is $\text{poly}(\ell) \cdot g(k) \cdot n^{o(1)}$ for some computable function g (where ℓ is the sensitivity parameter, i.e., the number of updates allowed per query). In the case of k-Path in directed graphs, and other graph problems, we specifically seek such an efficient sensitivity oracle in the *fully dynamic* setting where queries can change any ℓ edges by inserting and deleting them.

It is natural to ask that the query time has a *polynomial* dependence on ℓ (rather than, say, just a $f(\ell)$ dependence), as we do here, for two reasons. First, this is the dependence one would get by converting an efficient dynamic algorithm into a sensitivity oracle. Second, with our definition, any parameterized problem with an efficient sensitivity oracle is in FPT via the algorithm where the sensitivity oracle preprocesses an empty graph and then gets the full input graph as a query (but this would not be true if an arbitrary $f(\ell)$ term were allowed in the query time). Along the way to answering Question 1, we will also address more precisely the relationship between the classes of parameterized problems with efficient dynamic algorithms, efficient sensitivity oracles, and efficient static algorithms (a.k.a. the class FPT).

To our knowledge, such a *fully dynamic* notion of sensitivity oracles for parameterized problems has not been previously studied. The closest prior work is very recent [6] which considered a similar but *only decremental* setting, wherein queries may only delete edges from the graph, and not insert new edges. They design very elegant decremental sensitivity oracles for directed k-Path and for k-Vertex Cover, but their preprocessing and query times have exponential dependence on ℓ and hence are not "efficient" as we defined above. We also give evidence that the techniques of [6] cannot extend to the fully-dynamic setting; see Section 1.2 below for more details.

The second question we address is inspired by prior work on static algorithms for k-Path. Many fundamental techniques in the literature on parameterized algorithms were first introduced to study the k-Path problem. One such technique, algebraic coding (sometimes called "monomial testing" or "multilinear monomial detection"), is used in the current fastest static randomized algorithms for k-Path, and has also been used in other applications in algebraic complexity theory [22, 25, 10, 9, 8, 21]. Nonetheless, to our knowledge, these techniques have not been used in a dynamic or sensitivity setting before.

▶ **Question 2.** *Can algebraic coding techniques from the design of parameterized algorithms be used to design efficient dynamic algorithms or sensitivity oracles?*

A positive answer to Question 2 could lead to efficient dynamic algorithms or sensitivity oracles for a host of parameterized problems.

1.1 Our results

Let $\omega < 2.373$ be such that we can multiply two $n \times n$ matrices in $O(n^\omega)$ arithmetic operations [3]. Our first main result gives a positive answer to Question 1.

▶ **Theorem 3.** *The k-Path problem in directed graphs has an efficient parameterized sensitivity oracle. It can be solved with*[2]*:*

- *a Monte Carlo randomized algorithm with preprocessing time $2^k \text{poly}(k) n^\omega$ and query time $\ell^2 2^k \text{poly}(k)$, or*

[2] We work in the word-RAM model of computation with w-bit words for $w = O(\log n)$. Hence, only $O(\ell)$ words are needed to specify the ℓ edges to change in a query, and we can achieve query times independent of n.

- *a deterministic algorithm with preprocessing time $4^k \operatorname{poly}(k) n^\omega$ and query time $\ell^2 2^{\omega k}$.*

In addition to edge insertion and deletions, these algorithms also allow for vertex failures as part of the ℓ updates per query.

Although k-Path is known to not have an efficient dynamic parameterized algorithm (assuming the aforementioned 3SUM, Triangle, or Layered Reachability Oracle hardness assumptions from fine-grained complexity), Theorem 3 shows it does have an efficient parameterized sensitivity oracle. Since the reductions used in [2] are still valid in the sensitivity setting, one corollary is that the sensitivity versions of the 3SUM, Triangle, and Layered Reachability problems have efficient algorithms (although there are simple algorithms showing this for 3SUM and Triangle which do not go through k-Path; see Section 4 for more details).

It follows from prior work [18] that assuming another popular conjecture, the Strong Exponential Time Hypothesis (SETH), there are problems in FPT which do not have efficient parameterized sensitivity oracles (one example is the counting version of the single-source reachability problem; see again Section 4 for more details). Hence, assuming popular conjectures from fine-grained complexity, it follows that the class of parameterized problems with an efficient sensitivity oracle lies *strictly between* the class of parameterized problems with an efficient dynamic algorithm, and the class FPT (i.e., both class inclusions are strict).

Our sensitivity oracle for Theorem 3 uses $\Theta(n^2)$ space (for constant k), and it is natural to wonder whether this can be improved, especially since known dynamic parameterized algorithms for many other problems use much smaller space (e.g., many which dynamically maintain a small kernel [2]). However, we prove unconditionally that the space usage of our algorithm cannot be improved:

▶ **Theorem 4.** *Any randomized or deterministic sensitivity oracle for k-Path in directed graphs which handles edge insertion queries must use $\Omega(n^2)$ space.*

We also give three additional algorithmic results to complement Theorem 3. First, we extend Theorem 3 to give an algorithm for approximately counting k-paths:

▶ **Theorem 5.** *There is a randomized efficient parameterized sensitivity oracle which approximately counts the number of k-paths in an n-node, m-edge directed graph. For any $\epsilon > 0$, it produces an estimate to the number of k-paths in the graph that, with probability $> 99\%$, is within ϵ relative error, with preprocessing time $\epsilon^{-2} \cdot 4^k \operatorname{poly}(k) \cdot \min\{mn, n^\omega\}$ and update time $O\left(\epsilon^{-2} \cdot \ell^2 \cdot 2^{\omega k}\right)$. In addition to edge insertion and deletions, it also allows for vertex failures as part of the ℓ updates per query.*

Second, we present a randomized sensitivity oracle with a better dependence on k, at the cost of a worse dependence on ℓ, but only for undirected graphs:

▶ **Theorem 6** (Undirected graphs). *For the k-Path detection problem on an undirected graph G on n vertices, there exists a randomized sensitivity oracle with preprocessing time $O(1.66^k n^3)$ and query time $O(\ell^3 1.66^k)$.*

Third, we obtain the following corollary for bipartite graphs:

▶ **Corollary 7** (Undirected bipartite graphs). *For the k-Path detection problem on an undirected bipartite graph, where the partition of the vertices V into the two sides $V = S \cup T$ is known in advance and holds after any updates, there exists a sensitivity oracle with $2^{k/2} \operatorname{poly}(k) n^3$ preprocessing time and $\ell^3 2^{k/2} \operatorname{poly}(k)$ query time.*

Our algorithm for Theorem 3 uses a new dynamic version of the algebraic coding technique, and more specifically, a recent implementation called "extensor coding" by Brand, Dell, and Husfeldt [10]. See Section 2.5 and 3 below for more background on this technique, and the new ideas we introduce to be able to use it in a dynamic or sensitivity setting. We are then able to use and further modify our approach to design efficient dynamic parameterized algorithms for many problems for which no such algorithm was previously known, positively answering Question 2. We present the definitions for the problems we consider, followed by the results.

▶ **Definition 8** (k-Partial Cover). *Given a collection of subsets $S_1, ..., S_n \subseteq [N]$, find the minimum size of a sub-collection T of these, for which $\left|\bigcup_{S\in T} S\right| \geq k$, or declare that no such T exists.*

▶ **Definition 9** (m-Set k-Packing). *Given subsets $S_1, ..., S_n \subseteq [N]$, all with size $|S_i| = m$, decide whether there exists a sub-collection of k sets, which are all pairwise disjoint.*

▶ **Definition 10** (t-Dominating Set). *Given an undirected graph G on n vertices, find the minimum size of a set of vertices $S \subseteq V(G)$ such that $|S \cup N(S)| \geq t$ where $N(S)$ is the set of neighbors of vertices in S, i.e., $N(S) = \bigcup_{v\in S} N(v)$.*

▶ **Definition 11** (d-Dimensional k-Matching). *Fixing a universe $U = U_1 \times U_2 \times \ldots \times U_d$, where U_i are pairwise disjoint of combined size $|\biguplus_i U_i| = N$, and given a collection of tuples $T \subseteq U$, decide whether T contains a sub-collection of k pairwise disjoint tuples. (We say the tuples $a, b \in U$ are disjoint if $a[i] \neq b[i]$ for all $i = 1, 2, \ldots, d$.)*

▶ **Definition 12** (Exact k-Partial Cover). *Given a collection of subsets $S_1, ..., S_n \subseteq [N]$, decide whether there is a sub-collection T of these, in which all sets are pairwise-disjoint, and $\left|\biguplus_{S\in T} S\right| = k$.*

▶ **Theorem 13.** *There are efficient dynamic parameterized algorithms for the following problems. We write O^* here to hide factors that are polynomial in the parameters (m, k, t) and polylogarithmic in the size of the instance (n, N).*

- *For k-Partial Cover there are dynamic algorithms, with either randomized $O^*(2^k)$ or deterministic $O^*(4^k)$ update time.*
- *For m-Set k-Packing there are dynamic algorithms, with either randomized $O^*(2^{mk})$ or deterministic $O^*(4^{mk})$ update time.*
- *For t-Dominating Set there are dynamic algorithms, with either randomized $O^*(2^t)$ or deterministic $O^*(4^t)$ update time.*
- *For d-Dimensional k-Matching there are dynamic algorithms, with either randomized $O^*(2^{(d-1)k})$ or deterministic $O^*(4^{(d-1)k})$ update time.*
- *For Exact k-Partial Cover there are dynamic algorithms, with either randomized $O^*(2^k)$ or deterministic $O^*(4^k)$ update time.*

For the proof of Theorem 13 we refer the reader to the full version.

We also explain how the results can be extended to problems with additional constraints. As a simple example, we show how we can design an efficient sensitivity oracle for the problem of detecting whether a directed graph contains a walk of length k which visits at least $k-1$ vertices. As a second example, we discuss a different problem: given a directed graph G on n vertices, two (possibly intersecting) subsets $V_1, V_2 \subseteq V$, and two positive integers μ_1, μ_2, decide whether G contains a k-path that contains at most μ_1 vertices of V_1 and at most μ_2 vertices of V_2. For this problem we give a static deterministic algorithm running in time $4^{k+\min\{k,|V_1\cap V_2|\}}\operatorname{poly}(k)\cdot n^2$, as well as a sensitivity oracle counterpart. We refer the reader to the full version for the full discussion.

1.2 Comparison with related work

Algorithms for k-Path

The static k-Path problem has a long history. The current best upper bounds for it are a randomized $1.66^k\operatorname{poly}(n)$ algorithm in undirected graphs due to Björklund, Husfeldt, Kaski and Koivisto [8], a randomized time $2^k\operatorname{poly}(n)$ in directed (and hence also undirected) graphs due to Williams [26], and deterministic time $2.554^k\operatorname{poly}(n)$ in directed graphs due to Tsur [25].

In comparison, our randomized sensitivity oracle for directed graphs in Theorem 3 has a dependency on k (both in preprocessing and query time) of $O^*(2^k)$, so one cannot hope to improve on it without improving the best static algorithm. For undirected graphs, we give in Theorem 6 a sensitivity oracle with a dependency of $O^*(1.66^k)$, which matches the best bound of [8]. Our deterministic sensitivity oracle has a dependency of $O^*(4^k)$; we leave open the question of whether the techniques of [25] can be used in a sensitivity oracle setting to achieve $O^*(2.554^k)$.

There has also been work on the dynamic version of k-Path. However, Alman, Mnich and Vassilevska [2] showed that under the aforementioned fine-grained conjectures, there is no dynamic algorithm for *directed* graphs. At the same time, they give a deterministic dynamic algorithm for the case of *undirected* graphs, with update time $k!\cdot 2^{O(k)}\cdot\operatorname{polylog}(n)$. A subsequent result due to Chen et al. [11] presents a deterministic dynamic algorithm with amortized update time $2^{O(k^2)}$.

In comparison, our sensitivity oracle works for directed graphs just as well as for undirected graphs, and with a lower dependence on k (only $2^{O(k)}$). However, we obtain a sensitivity oracle, rather than a dynamic algorithm.

Decremental Parameterized Sensitivity Oracles

A recent paper by Bilò et al. [6] constructs decremental sensitivity oracles ("fault tolerant") for the k-Path problem (i.e., where all updates are edge deletions). They give one construction with a randomized $k^\ell 2^k\operatorname{poly}(n)$ preprocessing time and $O(\ell\min\{\ell,k+\log\ell\})$ update time, and a second construction with a lower preprocessing time of $2^k\cdot\frac{(\ell+k)^{\ell+k}}{\ell^\ell k^k}\cdot\ell\operatorname{poly}(n)$ at the expense of an increased query time of $O\left(\frac{(\ell+k)^{\ell+k}}{\ell^\ell k^k}\cdot\ell\min\{\ell,k\}\log n\right)$. They also give a deterministic algorithm, requiring $k^\ell 2.554^k\operatorname{poly}(n)$ preprocessing time and a very low $O(\ell\min\{\ell,k+\log\ell\})$ update time. Our Theorem 3 improves on the dependency on ℓ, making it polynomial in both the preprocessing and update times, and allows for both increments and decrements, while the methods of [6] are specific to decrements. We also achieve an update time which is independent of n in the word-RAM model. In [6], preprocessing is made aware of the value of ℓ, due to the superpolynomial dependence on it.

The techniques of Bilò et al. [6] also achieve a very low memory footprint for their fault-tolerant oracle, attaining at most logarithmic space dependency on n in all variants, when k and ℓ are considered constants. In contrast, we prove a lower bound, showing that one cannot hope for such dependency if increments are allowed, and that any fully dynamic sensitivity oracle must store at least $\Omega(n^2)$ bits (see Theorem 4). This suggests that the methods used in [6] cannot be used to devise a fully dynamic sensitivity oracle. Their algorithms are based on simple and elegant combinatorial arguments; for instance, in their algorithm with a faster preprocessing time, they precompute a collection of k-paths in the graph, in a way that ensures that with high probability, if there exists a path after ℓ edge deletions, one of the precomputed paths is still valid. We instead take a very different approach based on algebraic coding.

Cover, Matching, Dominating Set, and Packing problems

We are unaware of previous work on dynamic algorithms for the problems we study here (other than k-Path in undirected graphs, which we discussed above). Instead, we compare the dependence on k in the update times of our dynamic running times with the best known dependence on k for static algorithms. In the cases where we match, it is impossible to speed up the dependence on k in our dynamic algorithms without speeding up the fastest known static algorithms.

1. For both k-Partial Cover and t-Dominating Set, the fastest known static randomized running time is $2^k \operatorname{poly}(nk)$ by Koutis and Williams [22] (where for t-Dominating Set we replace k with t in the running time), which our dynamic algorithm matches, and the fastest known static deterministic running time is $2.554^k \operatorname{poly}(kn)$, stated by Tsur [25], whereas our deterministic dynamic algorithm achieves query time $O^*(4^k)$.
2. The fastest known static randomized algorithm for m-Set k-Packing by Björklund, Husfeldt, Kaski and Koivisto [8] has the intimidating running time of

   $$\left(\frac{0.108157 \cdot 2^m(1-1.64074/m)^{1.64076-m}m^{0.679625}}{(m-1)^{0.679623}}\right)^k n^6 \operatorname{poly}(N)$$

   This is less than $2^{mk}\operatorname{poly}(n)$, and considerably so for smaller m. For example, when $m=3$, the running time is bounded by $1.4953^{3k}n^6\operatorname{poly}(N)$. Due to the super-quadratic dependence on n, these techniques are unlikely to be adaptable for the dynamic case. In contrast, Koutis [20] proposes a static randomized algorithm running in time

   $$2^{mk} n \operatorname{poly}(mk)\operatorname{polylog}(n).$$

   Our dynamic result matches this running time.
3. The fastest known static randomized algorithm for d-Dimensional k-Matching, by Björklund, Husfeldt, Kaski, and Koivisto [8], runs in time $2^{(d-2)k}\operatorname{poly}(Nk)n$. In contrast, we match only an earlier algorithm by Koutis and Williams [22], which runs in time $2^{(d-1)k}\operatorname{poly}(kdn)$. The fastest known static deterministic algorithm runs in time $2.554^{(d-1)k}\operatorname{poly}(Nn)$, stated in Tsur [25], whereas we achieve deterministic dynamic update time $O^*(4^{(d-1)k})$.
4. Exact k-Partial Cover is a natural generalization of similar problems (such as m-Set k-Packing), though we are unaware of explicit prior work on it. A slight modification of the algorithm of Koutis [20] for m-Set k-Packing solves the static problem in randomized $O^*(2^k)$ time, matching the dependence on k in our dynamic algorithm.

Dynamic Parameterized Algorithms

Efficient dynamic algorithms have previously been proposed for a number of parameterized algorithms, often using dynamic versions of classic techniques from the design of fixed-parameter tractable algorithms. These techniques include dynamic kernels [2, 5, 12, 19], color-coding [2], dynamic elimination forests [11], dynamic branching trees [2], sketching [12], and other methods [14, 15]. To our knowledge, no previous work on dynamic parameterized algorithms has used algebraic techniques.

2 Preliminaries

2.1 Notation

For a positive integer k, write $[k] := \{1, 2, \dots, k\}$.

We will use the asymptotic notation $O^*(T)$; its meaning will depend on context, so we will define it each time it arises. Generally, in static algorithms and preprocessing times we hide factors that are polynomial in the input parameters (k, n, etc), but when talking about update times, we will only hide factors that are polylogarithmic in the input size.

Let ω be such that two $n \times n$ matrices can be multiplied using $O(n^\omega)$ field operations[3]; it is known that we can take $\omega < 2.373$ [3].

2.2 Sensitivity Oracles

▶ **Definition 14** (sensitivity oracle)**.** *A* sensitivity oracle *is a data structure containing two functions:*

1. *Initialization with an input instance of size n.*
2. *Query with ℓ changes (with problem-specific definition) to the* original input*. This returns the desired output on the altered input.*

The time spent in the initialization step is called the preprocessing time*, and the time spent in the query step is called the* query time*.*

In random sensitivity oracles, we assume the queries are independent of the randomness of the oracle, and thus cannot be used to fool a sensitivity oracle by obtaining the randomness used at the preprocessing phase. We say that a random sensitivity oracle errs with probability at most p, if for any pair of an initial configuration and a query, the probability of error is at most p.

We say that a sensitivity oracle for a parameterized problem, with parameter k, is *efficient* if its preprocessing time is $f(k)\,\mathrm{poly}(n)$ for some computable function f, and its query time is $\mathrm{poly}(\ell)g(k)n^{o(1)}$ for some computable function g.

2.3 Exterior algebra

A key technical tool we will make use of in our algorithms is the exterior algebra. We give a brief introduction to its definition and properties which we will use.

Given a field $\mathbb{F}$ and a positive integer k, the exterior algebra $\Lambda(\mathbb{F}^k)$ is a non-commutative ring over $\mathbb{F}$. In this paper we will be interested in the cases where $\mathbb{F}$ is either the rational numbers $\mathbb{Q}$, or a finite field $\mathbb{F}_{2^d}$ of characteristic 2.

Following the terminology of Brand, Dell and Husfeldt [10], elements of the exterior algebra are called *extensors*. Multiplication in $\Lambda(\mathbb{F}^k)$ is denoted by the wedge sign; for $x, y \in \Lambda(\mathbb{F}^k)$, their product is $x \wedge y$. The generators of this ring are $e_1, e_2, ..., e_k$, the standard basis of $\mathbb{F}^k$, and we impose the relations $e_i \wedge e_j = -e_j \wedge e_i$ for all $i, j \in [k]$ and $e_i \wedge e_i = 0$

[3] This is a slight abuse of notation. The exponent of matrix multiplication ω is typically defined as the smallest constant such that $n \times n$ matrices can be multiplied in $n^{\omega+o(1)}$ field operations, but we drop the "$o(1)$" in the exponent here for simplicity.

for all $i \in [k]$ (while the latter equation follows from the former for fields of characteristic different from 2, we still require it when $\operatorname{char} \mathbb{F} = 2$). We also require that the wedge product is bilinear[4]. Furthermore, it can be seen that the wedge product is associative[5].

Additionally, we consider any field element $a \in \mathbb{F}$ to be an element of $\Lambda(\mathbb{F}^k)$, with multiplication defined simply by $a \wedge e_i = e_i \wedge a = ae_i$.

For example, we have

$$\begin{aligned} e_1 \wedge (e_5 - e_2 + 3) \wedge e_3 &= e_1 \wedge e_5 \wedge e_3 - e_1 \wedge e_2 \wedge e_3 + 3e_1 \wedge e_3 \\ &= -e_1 \wedge e_3 \wedge e_5 - e_1 \wedge e_2 \wedge e_3 + 3e_1 \wedge e_3. \end{aligned}$$

To simplify the notation, for $I \subseteq [k]$ we also write $e_I := \bigwedge_{i\in I} e_i$, where the product is over the elements of I in ascending order, with the convention $e_\emptyset = 1$. We will sometimes also replace the wedge sign by a simple product sign, when it is clear from context that we mean the wedge product. For example, we can write $e_I := \prod_{i\in I} e_i$. Thus, the above example extensor can also be written as

$$e_1 \wedge (e_5 - e_2 + 3) \wedge e_3 = -e_{\{1,3,5\}} - e_{\{1,2,3\}} + 3e_{\{1,3\}}$$

We see that two products of the basis elements that differ only by the order of the terms can differ only in sign (e.g., $e_1 \wedge e_2 \wedge e_3 = -e_2 \wedge e_1 \wedge e_3$), and furthermore any product of e_is with a repeating factor vanishes (e.g., $e_1 \wedge e_2 \wedge e_2 = e_1 \wedge 0 = 0$). Thus, the set $\{e_I : I \subseteq [k]\}$ spans $\Lambda(\mathbb{F}^k)$ as a vector space over $\mathbb{F}$. It is in fact a basis of this vector space, and we have $\dim_{\mathbb{F}} \Lambda(\mathbb{F}^k) = 2^k$. That is, for any $x \in \Lambda(\mathbb{F}^k)$ there is a unique choice of the 2^k coefficients α_I so that $x = \sum_{I\subseteq[k]} \alpha_I e_I$. For any $x \in \Lambda(\mathbb{F}^k)$ and $T \subseteq [k]$ we denote by $[e_T]x$ the coefficient of e_T when x is represented in the basis $\{e_I : I \subseteq [k]\}$ (i.e., the value of α_T as above).

If there is a d such that all I for which $[e_I]x \neq 0$ have $|I| = d$, we say that x is a *degree-d* extensor. The space of degree-d extensors is denoted by $\Lambda^d(\mathbb{F}^k)$.

We identify any vector $v = (v[1], v[2], ..., v[k]) \in \mathbb{F}^k$ with the exterior algebra element $\sum_i v[i]e_i \in \Lambda^1(\mathbb{F}^k)$. A crucial property of the exterior algebra is that for any vector $v \in \mathbb{F}^k$ it holds that $v \wedge v = 0$. Indeed,

$$\begin{aligned} v \wedge v = \sum_{i,j} v[i]v[j]e_i \wedge e_j &= \sum_{i<j} v[i]v[j]e_i \wedge e_j + \sum_{i<j} v[i]v[j]e_j \wedge e_i \\ &= \sum_{i<j} v[i]v[j]e_i \wedge e_j - \sum_{i<j} v[i]v[j]e_i \wedge e_j = 0. \end{aligned}$$

Similarly, for any two vectors $u, v \in \mathbb{F}^k$, it holds that $u \wedge v = -v \wedge u$. It is important to notice that these properties do not hold for all elements in $\Lambda(\mathbb{F}^k)$. As an example, for $x = e_1 + e_2 \wedge e_3$ we have $x \wedge x = (e_1 + e_2 \wedge e_3) \wedge (e_1 + e_2 \wedge e_3) = e_1 \wedge e_2 \wedge e_3 + e_2 \wedge e_3 \wedge e_1 = 2e_1 \wedge e_2 \wedge e_3$. In fact, we can see that the wedge product of an even number of vectors commutes with any extensor.

Another very useful property of the exterior algebra is its connection to determinants. Let $v_1, v_2, ..., v_k \in \mathbb{F}^k$ be k vectors of dimension k, and consider their product $v_1 \wedge v_2 \wedge ... \wedge v_k$. Replace each v_i with its linear combination of the basis elements e_j and expand the product. We see that any monomial that repeats a basis element gets cancelled, and the others are all $e_{[k]}$ up to a sign. We get

$$v_1 \wedge v_2 \wedge ... \wedge v_k = \bigwedge_{i=1}^{k} \sum_{j=1}^{k} v_i[j]e_j = \sum_{\sigma\in S_k} \bigwedge_{i=1}^{k} v_i[\sigma(i)]e_{\sigma(i)} = \sum_{\sigma\in S_k} \operatorname{sgn}(\sigma)e_{[k]} \bigwedge_{i=1}^{k} v_i[\sigma(i)].$$

[4] $(a + b) \wedge c = (a \wedge c) + (b \wedge c)$ and $a \wedge (b + c) = (a \wedge b) + (a \wedge c)$.
[5] $(a \wedge b) \wedge c = a \wedge (b \wedge c)$

We recognize a determinant in the right hand side, thus showing that

$$v_1 \wedge v_2 \wedge ... \wedge v_k = \det(v_1|v_2|...|v_k)e_{[k]}. \tag{1}$$

2.4 Complexity of ring operations in the exterior algebra

We represent the elements of $\Lambda(\mathbb{F}^k)$ as the length-2^k vector of coefficients of the basis elements e_I. Addition is performed by coordinate-wise addition, hence requires $O(2^k)$ field operations. Multiplication is trickier, and there are a few important cases. All these cases were also noted and used in [10].

▶ **Proposition 15** (Skew product)**.** *Computing $x \wedge v$ for a general extensor $x \in \Lambda(\mathbb{F}^k)$ and a vector $v \in \mathbb{F}^k$ can be done in $O(2^k \cdot k)$ field operations.*

Proof sketch. This is accomplished by simply expanding the product. ◀

▶ **Proposition 16** (General product over characteristic 2)**.** *Computing $x \wedge y$ for any $x, y \in \Lambda(\mathbb{F}^k)$, when* $\mathrm{char}(\mathbb{F}) = 2$*, can be done in $O(2^k k^2)$ field operations.*

Proof sketch. Here, the ring $\Lambda(\mathbb{F}^k)$ is commutative. Then we can write $[e_I](x \wedge y) = \sum_{T \subseteq I}([e_T]x)([e_{I \setminus T}]y)$, which can be seen as a subset convolution operation. Using the fast subset convolution discovered by Björklund, Husfeldt, Kaski and Koivisto [7], we can compute this with $O(2^k k^2)$ field operations. ◀

▶ **Proposition 17** (General product over any characteristic)**.** *Computing $x \wedge y$ for $x, y \in \Lambda(\mathbb{F}^k)$, for $\mathbb{F}$ of any characteristic, can be done in $O(2^{\omega k/2})$ field operations.*

Proposition 17 is a result of Włodarczyk [27], which reduces computing $x \wedge y$ over $\Lambda(\mathbb{F}^k)$ to k^2 multiplications in a Clifford algebra, which in turn can be embedded in matrices of size $2^{k/2} \times 2^{k/2}$.

2.5 Extensor-coding

In this subsection, we give a brief overview of the techniques recently used by Brand, Dell and Husfeldt [10] to design a randomized and a deterministic algorithm for the (static) k-PATH problem; we heavily build off of these techniques in this paper.

Given a directed graph G, we denote its vertex set $V = \{v_1, ..., v_n\}$, and we denote by $W_s(G)$ the set of all walks in G of length s. Recall that walks may repeat vertices, whereas paths may not. For each edge $e \in E(G)$ we have an *edge variable* y_e, whose possible values will be in a field $\mathbb{F}$ we will pick later.

Furthermore, we define vectors $\chi : V \to \mathbb{F}^k$ by $\chi(v_i) = (1, j, j^2, \ldots, j^{k-1})$ where $j = f(i)$ for some injective function $f : [n] \to \mathbb{F}$. We call these *Vandermonde vectors.*

Given a walk in G of length s, $w = (w_1, w_2, ..., w_s) \in W_s(G)$, we define the corresponding *walk extensor* to be $\chi(w_1) \wedge y_{w_1 w_2} \wedge \chi(w_2) \wedge y_{w_2 w_3} \wedge \chi(w_3) \wedge ... \wedge y_{w_{s-1} w_s} \wedge \chi(w_s)$. We will sometimes denote the walk extensor of a walk w by $\chi(w)$.

Our goal is to compute the sum of all walk extensors for walks of length k,

$$\mathcal{Z} = \sum_{(w_1, w_2, \ldots, w_k) \in W_k(G)} \chi(w_1) \wedge y_{w_1 w_2} \wedge \chi(w_2) \wedge y_{w_2 w_3} \wedge \chi(w_3) \wedge ... \wedge y_{w_{k-1} w_k} \wedge \chi(w_k). \tag{2}$$

The result of this sum is seen to be proportional to the single basis element $e_{[k]}$, and by abuse of notation we will consider it as a scalar equal to that coefficient (or, more precisely, a polynomial in the y variables).

Any k-walk that is not a path does not contribute to Equation (2), because its walk extensor repeats a vector in the product. Thus, any nonzero term in the sum Equation (2) corresponds to a *k-path*.

The choice of $\chi(v_i)$ is such that for any walk $(w_1, w_2, ..., w_k)$ that is a k-path, the walk-extensor is a nonzero monomial in the y variables. This is seen using Equation (1), because $\chi(w_1) \wedge \chi(w_2) \wedge ... \wedge \chi(w_k) = e_{[k]} \det(\chi(w_1)|\chi(w_2)|...|\chi(w_k))$, which is the determinant of a Vandermonde matrix with unique columns, which is known to be nonzero.

Additionally, the monomial in the y variables given to each k-path is seen to uniquely identify the k-path (since there is a different y variable for each edge). Hence, the monomials of different k-paths cannot cancel, and we have proven:

▶ **Lemma 18.** *For a directed graph G, the value of $\mathcal{Z}$ given in Equation* (2) *is a nonzero polynomial in the y variables if and only if G contains a k-path.*

We now recall the classical DeMillo-Lipton-Schwartz-Zippel Lemma:

▶ **Lemma 19** ([13, 24, 28])**.** *Let $f \in \mathbb{F}[x_1, ..., x_n]$ be a nonzero polynomial in n variables with total degree at most d over some field $\mathbb{F}$, and let $S \subseteq \mathbb{F}$. For $a_1, a_2, ..., a_n \in S$ chosen uniformly at random, we have* $\Pr(f(a_1, a_2, ..., a_n) = 0) \leq \frac{d}{|S|}$.

Using the DeMillo-Lipton-Schwartz-Zippel Lemma, we now obtain a randomized algorithm for the k-Path detection problem, assuming we are able to efficiently compute $\mathcal{Z}$. Namely, for each $e \in E(G)$ we randomly select $y_e \in Y$ for some fixed $Y \subseteq \mathbb{F}$ of size $|Y| = 100k$, and compute $\mathcal{Z}$ as in Equation (2). If this value is nonzero, we know there is a k-path. Otherwise, there might still be a k-path, and we might have been unlucky and had the nonzero polynomial vanish for the specific choices of the y_e variables. We output that there is no k-path in this case. The probability of error is at most $\frac{k}{|Y|} = \frac{1}{100}$, and we can repeat this process to lower the probability of error as much as required.

It remains to show how to compute $\mathcal{Z}$ efficiently. We do this using dynamic programming. For any $1 \leq s \leq k$ and $1 \leq i \leq n$ we define $Q_s[i]$ as the sum of walk extensors of length s that end in v_i. Then $\mathcal{Z} = \sum_i Q_k[i]$, and we can compute the vector Q_{s+1} from Q_s by $Q_{s+1}[i] = \sum_{j:(v_j,v_i)\in E(G)} Q_s[j] \wedge y_{v_j,v_i} \wedge \chi(v_i)$. This requires kn^2 skew multiplications of extensors, each of which can be done in time $2^k \operatorname{poly}(k)$ (see Proposition 15). Thus, we can compute $\mathcal{Z}$ with $2^k \operatorname{poly}(k) n^2$ field operations, which is also the total running time of the randomized algorithm. We note that this works over any sufficiently large field $\mathbb{F}$ with $|\mathbb{F}| \geq 100k$.

Brand, Dell and Husfeldt [10] also give a deterministic variant of the algorithm, at the cost of increasing the time complexity. This is done with the beautiful idea of, for each vertex v, *"lifting"* the Vandermonde vector $\chi(v)$ to $\bar{\chi}(v) = \binom{\chi(v)}{0} \wedge \binom{0}{\chi(v)}$. By $\binom{\chi(v)}{0}$, we mean the vector of length $2k$ gotten by concatenating the vector $\chi(v)$ with k zeros. We then do the calculations in $\Lambda(\mathbb{F}^{2k})$ instead of $\Lambda(\mathbb{F}^k)$. Walk extensors of non-path walks still vanish. Now, as observed in [10], for any k-path we have

$$\bar{\chi}(w_1) \wedge \bar{\chi}(w_2) \wedge ... \wedge \bar{\chi}(w_k) = e_{[2k]} \det \left(\begin{pmatrix} \chi(w_1) \\ 0 \end{pmatrix} \middle| \begin{pmatrix} 0 \\ \chi(w_1) \end{pmatrix} ... \middle| \begin{pmatrix} \chi(w_k) \\ 0 \end{pmatrix} \middle| \begin{pmatrix} 0 \\ \chi(w_k) \end{pmatrix} \right)$$

By proper change of columns, the resulting determinant is equal to the determinant of a 2×2 block diagonal matrix, where the two diagonal blocks are identical. By basic properties of determinants, we then obtain

$$\bar{\chi}(w_1) \wedge \bar{\chi}(w_2) \wedge ... \wedge \bar{\chi}(w_k) = (-1)^{\binom{k}{2}} e_{[2k]} \det(\chi(w_1)|\chi(w_2)|...|\chi(w_k))^2. \tag{3}$$

Therefore, choosing $\mathbb{F} = \mathbb{Q}$, the walk extensors of different k-paths all have the same sign, and hence never cancel. The dynamic programming given above still works for this case, but extensor operations require $2^{2k} \operatorname{poly}(k)$ field operations (note we only need skew multiplications). Additionally, all numbers involved are of size at most $n^{O(k)} = 2^{O(k \log n)}$, and hence operations require only $\operatorname{poly}(k) \operatorname{polylog}(n)$ time, or $\operatorname{poly}(k)$ time in the word-RAM model which we assume. This produces a deterministic algorithm with running time $4^k \operatorname{poly}(k) n^2$.

3 Overview of our techniques

The key idea behind our algorithms is to identify appropriate sums of extensors which encode the answer to the problem (e.g., one where candidate solutions correspond to nonzero monomials), and which can be efficiently dynamically maintained. Similar to [10], we will frequently make use of the properties of exterior algebras to "nullify" repetitions (for example, of vertices in the k-Path problem, and of elements in the Exact k-Partial Cover problem). In many cases, the extensors or similar algebraic coding expressions used by the fastest known static algorithms seem difficult to efficiently maintain dynamically, and we will need to modify them, and identify useful precomputed values, to speed up our update time. While these techniques prove to be very well suited for designing sensitivity oracles and dynamic algorithms, we are unaware of previous work that uses the exterior algebra in such a way.

3.1 k-Path

For this problem, we aim to maintain $\mathcal{Z}$, the sum over walk extensors of length-k walks which we defined in Equation (2) above. Employing properties of the exterior algebra as discussed above, it is seen that any walk that is not a path (that is, a walk that repeats a vertex) does not contribute to the sum, and we thus indirectly compute a sum over only paths.

As we discussed in Section 2.5 above, Brand, Dell and Husfeldt [10] showed that computing this sum allows for extracting valuable information about the k-paths in a graph. In particular, by carefully selecting vectors or degree-2 extensors $\chi(v)$ for each vertex v, it allows for the design of randomized and deterministic algorithms for the k-path detection problem, as well as for approximately counting the number of k-paths with high probability.

Let us focus, first, on the case when our sensitivity oracle only allows for edge increments. We begin by precomputing, for each pair of vertices in the graph, a sum over the walk extensors between those vertices, so that we can "stitch" them together appropriately when new edges are inserted. Put precisely, let $W_s(u, v)$ denote the set of walks of length s from u to v, and let $Q_s[u, v]$ be the precomputed value of the sum of walk extensors for walks in $W_s(u, v)$ in the initial graph. Now suppose a new edge (v_1, v_2) is inserted. Then the additional walk extensors for walks of length s between any pair of vertices (t_1, t_2) can be computed as

$$\sum_{s'} \left(\sum_{w \in W_{s'}(t_1, v_1)} \chi(w) \right) \wedge y_{v_1, v_2} \wedge \left(\sum_{w \in W_{s-s'}(v_2, t_2)} \chi(w) \right)$$
$$= \sum_{s'} Q_{s'}[t_1, v_1] \wedge y_{v_1, v_2} \wedge Q_{s-s'}[v_2, t_2].$$

At first, stitching might seem problematic, since we might need to iterate over the possible lengths of each stitched part. This is not be a problem for a single edge insertion, but leads to an exponential dependency on the number of edges inserted as we partition the total length k between them. While this issue can be efficiently resolved with proper dynamic programming, we can circumvent this efficiently and more cleanly by instead working with sums over the walks of all possible lengths, while finally inspecting only the coefficient of $e_{[k]}$ in the resulting extensor, so that terms corresponding to paths of length other than k will either cancel out or have too low degree. Indeed, we note that only walks of length k contribute to this coefficient.

Another, more substantial problem occurs when one tries to "stitch" paths into a larger path that uses several new edges: there are $\ell!$ different orders to pass through the ℓ new edges. We combine properties of the exterior algebra with an algebraic trick (which we describe in more detail shortly) to allow computing the sum of the effects of all of these cases with only a polynomial dependence on ℓ.

In order to be able to work with only a small subset of the precomputed Q matrix, we additionally precompute the sum of walk extensors ending and beginning at each vertex (i.e., sums of rows and columns of Q), and the original sum over walk extensors in the original graph. This allows us to use only $O(\ell^2)$ precomputed values in the updating phase.

Finally, by further employing properties of the exterior algebra, we are able to combine these techniques with the inclusion-exclusion principle, to also allow edge removals, while keeping the same time and space complexities for preprocessing and updates.

We now describe the ideas in some more detail. We refer the reader to the full version for the full details. As a preprocessing, we compute an $n \times n$ matrix Q with extensor values, defined by

$$Q[i,j] = \sum_{\substack{(w_1,w_2,\ldots,w_s)\in W \\ w_1=v_i, w_s=v_j}} \chi(w_1) \wedge y_{w_1w_2} \wedge \chi(w_2) \wedge y_{w_2w_3} \wedge \chi(w_3) \wedge \ldots \wedge y_{w_{s-1}w_s} \wedge \chi(w_s)$$

where W is the set of *all walks* in the graph, which can have any length greater than 0. This is the sum of all walk extensors for walks from v_i to v_j. We then compute vectors S (resp. F) of length n, similarly computing in their ith entry the sum of walk extensors starting (resp. finishing) at each vertex v_i. That is, $S[i] = \sum_j Q[i,j]$ and $F[j] = \sum_i Q[i,j]$.

Finally, we compute $\mathcal{Z} = \sum_{i,j} Q[i,j]$, the sum of all walk extensors over all walks. $e_{[k]}\mathcal{Z}$ is exactly the value we aim to maintain after a query, as this is exactly the one used in the extensor coding technique (Section 2.5). We explain how with proper dynamic programming we can compute these values in preprocessing with $2^k \operatorname{poly}(k)n^2$ operations over $\mathbb{F}$, using only additions and skew multiplications.

Then, when prompted with a query, we aim to compute the sum over walk extensors in the updated graph, denoted $\mathcal{Z}_{\text{new}}$. Inspecting whether $e_{[k]}\mathcal{Z}_{\text{new}} \neq 0$ will let us test whether the updated graph contains a k-path.

We now begin handling a query. Given a list of ℓ updates that are each either an edge insertion or edge deletion, there are $n' \leq 2\ell$ vertices at the endpoints of the updated edges. We begin by extracting the $n' \times n'$ sub-matrix Q' of Q, and length-n' sub-vectors S', F' of S, F, gotten by taking entries corresponding to those endpoint vertices.

We next define an $n' \times n'$ matrix E_r^+ corresponding to the r-th edge insertion by setting each of its entries to 0 except for $E_r^+[i,j] = y_{i,j}$, where (v_i, v_j) is the r-th inserted edge. We similarly define E_r^- in the same way, to be the 0 matrix except for $E_r^-[i,j] = y_{i,j}$ where (v_i, v_j) is the r-th deleted edge. We then define $\Delta^+ = \sum_r E_r^+$, $\Delta^- = \sum_r E_r^-$, and $\Delta = \Delta^+ - \Delta^-$. We then compute that

$$\mathcal{Z}_{\text{new}} = \mathcal{Z} + \sum_{i=1}^{k} F'^T \Delta (Q'\Delta)^{i-1} S'.$$

This formula is crucial to our query algorithm, so we explain it in some detail here.

We first explain the correctness of this formula for the case of only increments. In this case, $\mathcal{Z}_{\text{new}} - \mathcal{Z}$ is exactly the sum of walk-extensors for walks that use the new edges. Here, upon expanding the expression when $\sum_r E_r^+$ is substituted for Δ^+, we note that $F'^T\Delta^+S'$ counts the walks that use exactly one of the new edges. Indeed, for each r the term $F'^T E_r^+ S'$ accounts for walks passing through the r-th new edge exactly once. For counting walks that use exactly two of the newly inserted edges, we now need to compute $\sum_{r_1 \neq r_2} F'^T E_{r_1}^+ Q' E_{r_2}^+ S'$. Indeed, this expression correctly accounts for walks with nonzero walk extensors starting at any vertex, continuing through one new edge, travelling to a new edge then through it, then continuing to end at any vertex. We further note that any specific path is counted in this way exactly once.

We note that by linearity and the fact that only paths produce nonzero walk extensors, this is equal to $F'^T\Delta^+Q'\Delta^+S'$. That is, upon expanding the expression when $\sum_r E_r^+$ is substituted for Δ^+, we get exactly the terms we intended, plus terms accounting for walks passing through the new edges twice, which evaluate to zero in the exterior algebra. This considerably simplifies the calculations. Continuing in the same fashion, we argue we obtain

$$\mathcal{Z}_{\text{new}} - \mathcal{Z} = \sum_{i=1}^{k} F'^T\Delta^+(Q'\Delta^+)^{i-1}S'$$

in total for the incremental case.

Consider now the general case, with both increments and decrements. Suppose w is any walk on the vertices which uses edges from the union of the original graph and the updated graph. We assume w is a path, since otherwise its walk extensor vanishes, and we need not worry about whether it appears in the sum $\mathcal{Z}_{\text{new}}$. Now suppose w has length at most k and uses a inserted edges and b removed edges.

Consider first when $a \geq 1$. In this case, w is not counted in the original sum $\mathcal{Z}$. When substituting $\Delta = \sum_j E_j^+ - \sum_j E_j^-$ into $\sum_{i=1}^k F'^T\Delta(Q'\Delta)^{i-1}S'$ and expanding, we see that w is counted only when all the chosen E_j^+ factors exactly correspond to the a new edges used by w, in the order they are used. It is also counted when choosing any $b' \leq b$ factors of type E_j^- that appear in w, but they must also appear in the right order, and they contribute the walk extensor of w with weight exactly $(-1)^{b'}$. In total, w is accounted for exactly $\sum_{b'=0}^{b} \binom{b}{b'}(-1)^{b'} = (1-1)^b = [b = 0]$ times[6] in $\mathcal{Z}_{\text{new}}$, which is what we want.

Now suppose $a = 0$. Then a similar argument shows that the number of times it is counted in $\sum_{i=1}^k F'^T\Delta(Q'\Delta)^{i-1}S'$ is exactly $\sum_{b'=1}^{b} \binom{b}{b'}(-1)^{b'} = (1-1)^b - 1 = [b = 0] - 1$. However, it is also counted in $\mathcal{Z}$ exactly once, so in total is counted $[b = 0]$ times, which is also exactly what we want.

Using this formula, we explain how we are generally able to compute $\mathcal{Z}_{\text{new}}$ in $O(\ell^2 2^{\omega k/2})$ field operations. However, we come back to this running time once we describe the specifics of the randomized and deterministic sensitivity oracles, each with its own details.

3.2 k-Partial Cover

We also make use of extensors to design fully dynamic algorithms for other parameterized problems. We focus here on one example: designing a deterministic dynamic algorithm for the k-Partial Cover problem. In this problem, we are given subsets $S_1, ..., S_n \subseteq [N]$, and wish to find the minimum number of such subsets whose union has size at least k (or report that no such collection exists).

[6] Here we use the notation $[b = 0] := \begin{cases} 1 & \text{if } b = 0, \\ 0 & \text{otherwise.} \end{cases}$

One could use a polynomial constructed by Koutis and Williams [22] for this problem, combined with similar techniques to those we used above for k-PATH, to devise a deterministic dynamic algorithm with update time $O(2^{\omega k})$. However, we instead give a different polynomial which is easier to dynamically maintain, achieving a faster $O^*(4^k)$ update time.

As in the k-PATH problem, we give each element in the universe $a \in [N]$ a Vandermonde vector $\chi(a) \in \mathbb{Q}^k$, then lift it to $\bar{\chi}(a) = \binom{\chi(a)}{0} \wedge \binom{0}{\chi(a)} \in \Lambda(\mathbb{Q}^{2k})$.

We introduce a single new variable z, and consider the polynomial

$$P(z) = \prod_S \left(1 + z\left[\prod_{a\in S}(1+\bar{\chi}(a)) - 1\right]\right).$$

We argue that the solution we seek is the minimum t for which $[e_{[2k]}z^t]P \neq 0$ (that is, the coefficient of $e_{[2k]}$ in the coefficient of z^t in P). Our goal, then, will be to maintain the value of $P(z)$, and calculations will take place in polynomials over extensors, $\Lambda(\mathbb{Q}^{2k})[z]$. We also argue that $\deg P(z) \leq k$, and so is not too large to handle in update steps.

We first argue that the product in P should only be computed over sets S of cardinality less than k, as larger sets can be treated separately, knowing that the optimal solution is 1 if any such set exists in the collection. Since $P(z)$ is defined as a product over sets, to update it with a new set S of cardinality $|S| < k$, we can multiply $P(z)$ by the corresponding factor $1+z\left[\prod_{a\in S}(1+\bar{\chi}(a)) - 1\right]$. This seems too slow at first, since general extensor multiplication requires $O(2^{\omega k})$ time, which is more than our target $O^*(4^k)$. However, rather than computing the factor and then using general extensor multiplication, we argue that we can indirectly multiply by this factor using only additions and skew multiplications by rearranging the contributing terms, thus requiring only $O^*(4^k)$ field operations. Indeed, the product of $P(z)$ with the new factor is $(1-z)P(z) + P(z)\prod_{a\in S}(1+\bar{\chi}(a))$, which can be performed by separately computing $(1-z)P(z)$ and $P(z)\prod_{a\in S}(1+\bar{\chi}(a))$, where the latter can be computed by repeated skew-multiplications.

A new problem arises when a set S is removed. For this, we need to somehow cancel the factor in $P(z)$ corresponding to S. We show that in this case the corresponding factor has an inverse in $\Lambda(\mathbb{Q}^{2k})[z]$: we first note that it can be written as $1 + X$ for an element X that contains only extensors of degree ≥ 2, which implies that $X^{k+1} = 0$, and using this, we observe that

$$(1+X)(1-X+X^2-\ldots+(-X)^k) = 1-(-X)^{k+1} = 1,$$

so $(1+X)^{-1} = 1-X+X^2-\ldots+(-X)^k$. Similar to the previous case, we argue that multiplying by this factor can also be done with $\mathrm{poly}(k)$ additions and skew multiplications, and so can be done in $O^*(4^k)$ time.

This and all our other dynamic algorithms are described in detail in the full version.

4 Discussion on fixed-parameter complexity classes

We discuss the relationship between the following definitions.

▶ **Definition 20** (FPT). *A parameterized problem is in FPT if it is decidable in time $f(k)\,\mathrm{poly}(n)$ for a computable function f.*

▶ **Definition 21** (FPD). *A parameterized problem is in FPD (Fixed-Parameter Dynamic) if there is a dynamic algorithm for it requiring $f(k)\,\mathrm{poly}(n)$ preprocessing time and $g(k)n^{o(1)}$ update time, for computable functions f and g.*

▶ **Definition 22** (FPSO). *A parameterized problem is in FPSO (Fixed-Parameter Sensitivity Oracle) if there is a sensitivity oracle for it requiring $f(k)\,\mathrm{poly}(n)$ preprocessing time and $\mathrm{poly}(\ell)g(k)n^{o(1)}$ update time, for a computable function g.*

We note that FPD $\subseteq$ FPSO $\subseteq$ FPT. We can show that the inclusions are strict, at least under plausible hardness conjectures. For example, [2] shows that k-Path in directed graphs does not admit a dynamic algorithm under hardness conjectures (which are shown in this paper to be in FPSO). [18] shows that the #SSR problem does not have an efficient sensitivity oracle, assuming SETH, which in turn can be used to show that, assuming SETH, there exists an FPT problem that is not in FPSO.

We further note that many problems shown in [2] to not be in FPD can be shown to be in FPSO. As a few examples:

1. Triangle Detection, which is the problem of detecting whether a graph contains a triangle, can be solved efficiently by a sensitivity oracle by precomputing the square of the graph's adjacency matrix and computing the number of triangles in time $O(n^\omega)$, then updating the number of triangles in time $O(\ell^\omega)$ ($O(\ell)$ for triangles that use a single updated edge, $O(\ell^2)$ for triangles using two updated edges, and $O(\ell^\omega)$ for those using three updated edges).
2. Incremental st-Reachability, which is the problem of deciding whether two predetermined vertices s and t are connected in a directed graph, only allowing incremental updates, can be solved efficiently by precomputing reachability between any two vertices (for example by running BFS from all vertices in time $\mathrm{poly}(n)$), then using dynamic programming to answer a query in time $\mathrm{poly}(\ell)$ by updating the connectivity information only on the $\leq 2\ell + 2$ vertices that are either s, t or are part of any inserted edge
3. 3SUM, which is the problem of deciding whether there are 3 elements in a list that sum to 0. Here it is possible to precompute the sums of all pairs in time $O(n^2)$ (counting multiplicities), and the number of triples whose sum is 0. Then when adding or removing ℓ numbers it is possible to compute in $\mathrm{poly}(\ell)$ time the number of new solutions and the number of previous solutions that should be removed, and checking if the remaining number of solutions is nonzero.
4. k-Layered Reachability Oracle (k-LRO), the problem of deciding whether two vertices u, v are connected in a directed k-layered graph (that is, a graph whose vertices are partitioned into k parts, with edges only going from one part to the next), also has an efficient sensitivity oracle, and in fact can be seen to be equivalent to the directed k-Path problem. In particular, the same k-Path sensitivity oracle devised in this paper can be used here without any changes.

5 Open questions

We briefly discuss a few natural questions that arise.

1. Is there a fully-dynamic k-Path detection algorithm on undirected graphs with $f(k)n^{O(1)}$ preprocessing time for some computable function f, and update time $2^{O(k)}n^{o(1)}$?
   This would beat the dynamic algorithm proposed in [2] that is a simple adaptation of the original color-coding idea [4], while to the best of our knowledge, no improvement on this original color-coding idea is known to transfer to the dynamic setting. While our work shows a better dependency on k, we do not reach a truly dynamic algorithm, but rather only a sensitivity oracle.
   We note that it is unlikely that such an an algorithm exists for directed graphs, due to the conditional lower bound presented in [2].

2. Is there an efficient sensitivity oracle for the k-TREE problem?
   The k-TREE problem is as follows: given an undirected graph G on n vertices and a tree T on k vertices, determine whether T is a (not necessarily induced) subgraph of G. This problem is known to be FPT, and for example is shown in [22] to be solvable in time $2^k \operatorname{poly}(k) \operatorname{poly}(n)$, with techniques similar to those applied for the k-PATH problem. However, it is unclear how to adapt the techniques here for an efficient sensitivity oracle. We conjecture that there is, in fact, no efficient sensitivity oracle for this problem. More specifically, we conjecture this for any k-TREE formed by connecting a single vertex to the beginning of $\Theta(\sqrt{k})$ paths of length $\Theta(\sqrt{k})$. We note that the techniques of [6] for a decremental sensitivity oracle work for any k-TREE just as well as they do for the k-PATH problem.
3. Brand [9] noted that the algebra generated by the lifts of Vandermonde vectors, which is the algebra used in all the deterministic algorithms presented in this paper, has dimension $O(\varphi^{2k})$ where $\varphi = \frac{1+\sqrt{5}}{2}$ - much smaller than the anticipated $O(4^k)$. It is also commutative. It is an open problem whether, in light of this, multiplication in this algebra can be reduced to below $O(2^{\omega k})$. Such an algorithm will immediately improve the bounds discussed in this paper.
4. Can other techniques used to solve the static versions of the problems discussed in this paper, or other parameterized problems, be used to design faster dynamic algorithms and sensitivity oracles?

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 434–443. IEEE, 2014.

2 Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic parameterized problems and algorithms. *ACM Trans. Algorithms*, 16(4), July 2020. doi:10.1145/3395037.

3 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '21, pages 522–539, USA, 2021. Society for Industrial and Applied Mathematics. URL: https://dl.acm.org/doi/10.5555/3458064.3458096.

4 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995. doi:10.1145/210332.210337.

5 Max Bannach, Zacharias Heinrich, Rüdiger Reischuk, and Till Tantau. Dynamic Kernels for Hitting Sets and Set Packing. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2021.7.

6 Davide Bilò, Katrin Casel, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, J.A. Gregor Lagodzinski, Martin Schirneck, and Simon Wietheger. Fixed-Parameter Sensitivity Oracles. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:18, 2022. doi:10.4230/LIPIcs.ITCS.2022.23.

7 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: Fast subset convolution. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 67–74, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1250790.1250801.

8 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *Journal of Computer and System Sciences*, 87:119–139, 2017. doi:10.1016/j.jcss.2017.03.003.

9 Cornelius Brand. Patching Colors with Tensors. In *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:16, 2019. doi:10.4230/LIPIcs.ESA.2019.25.

10 Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 151–164, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188902.

11 Jiehua Chen, Wojciech Czerwiński, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michał Pilipczuk, Manuel Sorge, Bartłomiej Wróblewski, et al. Efficient fully dynamic elimination forests with applications to detecting long paths and cycles. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 796–809. SIAM, 2021. doi:10.1137/1.9781611976465.50.

12 Rajesh Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *Proceedings of the 2015 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1234–1251, 2015. doi:10.1137/1.9781611973730.82.

13 Richard A. Demillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978. doi:10.1016/0020-0190(78)90067-4.

14 Zdeněk Dvořák, Martin Kupec, and Vojtěch Tůma. A dynamic data structure for mso properties in graphs with bounded tree-depth. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014*, pages 334–345, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. doi:10.1007/978-3-662-44777-2_28.

15 Zdeněk Dvořák and Vojtěch Tůma. A dynamic data structure for counting subgraphs in sparse graphs. In *Proceedings of the 13th International Conference on Algorithms and Data Structures*, WADS'13, pages 304–315, Berlin, Heidelberg, 2013. Springer-Verlag. doi:10.1007/978-3-642-40104-6_27.

16 Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms. *CoRR*, abs/2102.11169, 2021. arXiv:2102.11169.

17 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30, 2015.

18 Monika Henzinger, Andrea Lincoln, Stefan Neumann, and Virginia Vassilevska Williams. Conditional hardness for sensitivity problems. *CoRR*, 2017. arXiv:1703.01638.

19 Yoichi Iwata and Keigo Oka. Fast dynamic graph algorithms for parameterized problems. *ArXiv*, abs/1404.7307, 2014. arXiv:1404.7307.

20 Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Automata, Languages and Programming*, pages 575–586, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-70575-8_47.

21 Ioannis Koutis. Constrained multilinear detection for faster functional motif discovery. *Information Processing Letters*, 112(22):889–892, November 2012. doi:10.1016/j.ipl.2012.08.008.

22 Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. *ACM Trans. Algorithms*, 12(3), May 2016. doi:10.1145/2885499.

23 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 603–610, 2010.

24 J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980. doi:10.1145/322217.322225.

25 Dekel Tsur. Faster deterministic parameterized algorithm for k-path. *Theoretical Computer Science*, 790:96–104, 2019. doi:10.1016/j.tcs.2019.04.024.

26 Ryan Williams. Finding paths of length k in $\mathcal{O}^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, February 2009. doi:10.1016/j.ipl.2008.11.004.

27 Michał Włodarczyk. Clifford algebras meet tree decompositions. *Algorithmica*, 81(2):497–518, February 2019. doi:10.1007/s00453-018-0489-3.

28 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*, EUROSAM '79, pages 216–226. Springer-Verlag, 1979. doi:10.5555/646670.698972.