ALGORITHMIC MANIPULATION OF PROBABILITY DISTRIBUTIONS FOR NETWORKS AND MECHANISMS

A Dissertation
Presented to
The Academic Faculty

By

David Durfee

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology

May 2019

ALGORITHMIC MANIPULATION OF PROBABILITY DISTRIBUTIONS FOR NETWORKS AND MECHANISMS

Approved by:

Dr. Richard Peng, Advisor
School of Computer Science
Georgia Institute of Technology

Dr. Santosh Vempala
School of Computer Science
Georgia Institute of Technology

Dr. Xi Chen
School of Computer Science
Columbia University

Dr. Eric Vigoda
School of Computer Science
Georgia Institute of Technology

Dr. Alejandro Toriello
School of Industrial Systems and Engineering
Georgia Institute of Technology

Date Approved: December , 2018

ACKNOWLEDGEMENTS

The most thanks must go to my advisor, Richard Peng, a one-of-a-kind researcher and one of the hardest working people I've ever met. He was always involved in many different projects, and yet still managed to be incredibly generous with his time. His abundance of interesting open problems and new approaches, combined with his unprecedented receptiveness to collaboration, is truly exceptional.

I would also like to especially thank Xi Chen for being instrumental to the beginning of my graduate research.

Thanks to all coauthors I've had the pleasure of working with, and to the members of my thesis committee, particularly Santosh Vempala for being the reader of my thesis. Thanks to friends and fellow grad students at Georgia Tech.

Lastly, I would like to of course thank my family, and especially my parents for always helping, encouraging, and putting up with any endeavor I have pursued throughout my life.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

In this thesis we present four different works that solve problems in dynamic graph algorithms, spectral graph algorithms, computational economics, and differential privacy. While these areas are not all strongly correlated, there were similar techniques integral to each of the results. In particular, a key to each result was carefully constructing probability distributions that interact with fast algorithms on networks or mechanisms for economic games and private data output. For the fast algorithms on networks this required utilizing essential graph properties for each network to determine sampling probabilities for sparsification procedures that we often recursively applied to achieve runtime speedups. For mechanisms in economic games we construct a gadget game mechanism by carefully manipulating the expected payoff resulting from the probability distribution on the strategy space to give a correspondence between two economic games and imply a hardness equivalence. For mechanisms on private data output we construct a smoothing framework for input data that allows private output from known mechanisms while still maintaining certain levels of accuracy.

Dynamic Spectral Sparsification In [1], we consider a dynamically changing graph under edge insertions and deletions, and give a data structure for maintaining a $(1 \pm \epsilon)$-cut sparsifier in worst-case update time $poly(\log n, \epsilon^{-1})$, and a $(1 \pm \epsilon)$-spectral sparsifier in amortized update time $poly(\log n, \epsilon^{-1})$. We also developed a vertex sparsification routine, that improves upon [2], which samples vertices according to a distribution carefully obtained by considering the connectivity properties of the Schur complement that results from eliminating a set of independent vertices. We then combined our data structures and vertex sparsification routine to maintain a $(1 - \epsilon)$ approximate max-flow in undirected, unweighted bipartite graphs with amortized update time $poly(\log n, \epsilon^{-1})$.

Determinant-preserving Sparsification In [3] we construct a specific edge sampling distribution using leverage scores to approximately maintain determinant of the minor of a Laplacian matrix, a more delicate quantity to maintain compared to spectral approximation. The proof utilizes the connection between determinant and spanning trees established by Kirchhoff's matrix tree theorem, along with extending a concentration bound in [4]. We then incorporate this sparsification procedure into a fast and sparse Schur complement routine by further refining our sampling distribution with random walks, which allows for the use of recursive algorithms. Using connections between Schur complement and determinant, we give an $\tilde{O}(n^2\delta^{-2})$-time algorithm for computing a $(1 \pm \delta)$-approximate determinant of the Laplacian minor. This is the first routine for graphs that outperforms general-purpose routines for computing determinants of arbitrary matrices. This general structure can also be used to output a random spanning tree in $\tilde{O}(n^2\delta^{-2})$-time from a distribution that has total variation distance $\leq \delta$ from the true distribution.

Computational Equilibria Hardness In [5], we proved that computing an equilibrium for an important class of games, anonymous games, is PPAD-complete, confirming a conjecture by Daskalakis and Papadimitriou put forth after a series of papers on anonymous games [6, 7, 8, 9]. In order to achieve the hardness reduction we considered a known PPAD-hard class of games, polymatrix games, and constructed an anonymous gadget game that gave a correspondence between the equilibria of each game. This correspondence was obtained by carefully tuning the expected payoffs resulting from the probability distribution determined by our constructed gadget game, resulting in the key lemma to our reduction.

Sensitivity Preprocessing for Privacy In [10], we give a recursive function preprocessing routine to smooth the output probability distribution from applying standard differential privacy mechanisms. A variety of techniques have been used for a similar purpose such as smooth sensitivity and Sample-and-Aggregate [11], Propose-Test-Release [12], and Lipschitz extensions [13, 14, 15]. Our framework is most similar to Lipschitz extensions,

but overcomes some of the limitations of the previous techniques and works in a more generalized setting. In particular, using certain probability tricks we are further able to efficiently implement our recursion in $O(n^2)$ time for important statistical metrics such as mean and variance, neither of which were achievable with the previous techniques. Additionally, we extend our framework to a more refined smoothness measure and show how this can serve as a useful tool for a variant privacy definition and its applications in markets for privacy.

CHAPTER 1
ON FULLY DYNAMIC GRAPH SPARSIFIERS

This was joint work with Ittai Abraham, Ioannis Koutis, Sebastian Krinninger, and Richard Peng.

1.1 Abstract

We initiate the study of fast dynamic algorithms for graph sparsification problems and obtain fully dynamic algorithms, allowing both edge insertions and edge deletions, that take polylogarithmic time after each update in the graph. Our three main results are as follows. First, we give a fully dynamic algorithm for maintaining a $(1 \pm \epsilon)$-spectral sparsifier with amortized update time $\mathrm{poly}(\log n, \epsilon^{-1})$. Second, we give a fully dynamic algorithm for maintaining a $(1 \pm \epsilon)$-cut sparsifier with *worst-case* update time $\mathrm{poly}(\log n, \epsilon^{-1})$. Both sparsifiers have size $n \cdot \mathrm{poly}(\log n, \epsilon^{-1})$. Third, we apply our dynamic sparsifier algorithm to obtain a fully dynamic algorithm for maintaining a $(1 + \epsilon)$-approximation to the value of the maximum flow in an unweighted, undirected, bipartite graph with amortized update time $\mathrm{poly}(\log n, \epsilon^{-1})$.

1.2 Introduction

Problems motivated by graph cuts are well studied in theory and practice. The prevalence of large graphs motivated sublinear time algorithms for cut based problems such as clustering [16, 17, 18, 19, 20, 21]. In many cases such as social networks or road networks, these algorithms need to run on dynamically evolving graphs. In this paper, we study an approach for obtaining sublinear time algorithms for these problems based on dynamically maintaining graph sparsifiers.

Recent years have seen a surge of interest in dynamic graph algorithms. On the one hand, very efficient algorithms, with polylogarithmic running time per update in the graph, could be found for some key problems in the field [22, 23, 24, 25, 26, 27, 28, 29]. On the other hand, there are polynomial conditional lower bounds for many basic graph problems [30, 31, 32]. This leads to the question which problems can be solved with polylogarithmic update time. Another relatively recent trend in graph algorithmics is graph sparsification where we reduce the size of graphs while approximately preserving key properties such as the sizes of cuts [33]. These routines and their extensions to the spectral setting [34, 35] play central roles in a number of recent algorithmic advances [36, 37, 38, 39, 40, 41, 42], often leading to graph algorithms that run in almost-linear time. In this paper, we study problems at the intersection of dynamic algorithms and graph sparsification, leveraging ideas from both fields.

At the core of our approach are data structures that dynamically maintain graph sparsifiers in $\mathrm{polylog}\, n$ time per edge insertion or deletion. They are motivated by the spanner based constructions of spectral sparsifiers of Koutis [43]. By modifying dynamic algorithms for spanners [29], we obtain data structures that spend amortized $\mathrm{polylog}\, n$ per update. Our main result for spectral sparsifiers is:

Theorem 1.2.1. *Given a graph with polynomially bounded edge weights, we can dynamically maintain a* $(1 \pm \epsilon)$*-spectral sparsifier of size* $n \cdot \mathrm{poly}(\log n, \epsilon^{-1})$ *with amortized update time* $\mathrm{poly}(\log n, \epsilon^{-1})$ *per edge insertion / deletion.*

When used as a black box, this routine allows us to run cut algorithms on sparse graphs instead of the original, denser network. Its guarantees interact well with most routines that compute minimum cuts or solve linear systems in the graph Laplacian. Some of them include:

1. min-cuts, sparsest cuts, and separators [44],

2. eigenvector and heat kernel computations [45],

3. approximate Lipschitz learning on graphs [46] and a variety of matrix polynomials in the graph Laplacian [47].

In many applications the full power of spectral sparsifiers is not needed, and it suffices to work with a cut sparsifier. As spectral approximations imply cut approximations, research in recent years has focused spectral sparsification algorithms [48, 49, 50, 51, 52, 53]. In the dynamic setting however we get a strictly stronger result for cut sparsifiers than for spectral sparsifiers: we can dynamically maintain cut sparsifiers with polylogarithmic *worst-case* update time after each insertion / deletion. We achieve this by generalizing Koutis' sparsification paradigm [43] and replacing spanners with approximate maximum spanning trees in the construction. While there are no non-trivial results for maintaining spanners with worst-case update time, spanning trees can be maintained with polylogarithmic worst-case update time by a recent breakthrough result [24]. This allows us to obtain the following result for cut sparsifiers:

Theorem 1.2.2. *Given a graph with polynomially bounded edge weights, we can dynamically maintain a* $(1 \pm \epsilon)$*-cut sparsifier of size* $n \cdot \mathrm{poly}(\log n, \epsilon^{-1})$ *with worst-case update time* $\mathrm{poly}(\log n, \epsilon^{-1})$ *per edge insertion / deletion.*

We then explore more sophisticated applications of dynamic graph sparsifiers. A key property of these sparsifiers is that they have arboricity $\mathrm{polylog}\, n$. This means the sparsifier is locally sparse, and can be represented as a union of spanning trees. This property is becoming increasingly important in recent works [26, 54]: Peleg and Solomon [54] gave data structures for maintaining approximate maximum matchings on fully dynamic graphs with amortized cost parameterized by the arboricity of the graphs. We demonstrate the applicability of our data structures for designing better data structures on the undirected variant of the problem. Through a two-stage application of graph sparsifiers, we obtain the first non-separator based approach for dynamically maintaining $(1 - \epsilon)$-approximate maximum flow on fully dynamic graphs:

Theorem 1.2.3. *Given a dynamically changing unweighted, undirected, bipartite graph $G = (A, B, E)$ with demand -1 on every vertex in A and demand 1 on every vertex in B, we can maintain a $(1 - \epsilon)$-approximation to the value of the maximum flow, as well as query access to the associated approximate minimum cut, with amortized update time $\mathrm{poly}(\log n, \epsilon^{-1})$ per edge insertion / deletion.*

To obtain this result we give stronger guarantees for vertex sparsification in bipartite graphs, identical to the terminal cut sparsifier question addressed by Andoni, Gupta, and Krauthgamer [2]. Our new analysis profits from the ideas we develop by going back and forth between combinatorial reductions and spectral sparsification. This allows us to analyze a vertex sampling process via a mirror edge sampling process, which is in turn much better understood.

Overall, our algorithms bring together a wide range of tools from data structures, spanners, and randomized algorithms. We will provide more details on our routines, as well as how they relate to existing combinatorial and probabilistic tools in Section 2.4.

1.3 Background

1.3.1 Dynamic Graph Algorithms

In this paper we consider undirected graphs $G = (V, E)$ with n vertices and m edges that are either unweighted or have non-negative edge weights. We denote the weight of an edge $e = (u, v)$ in a graph G by $w_G(e)$ or $w_G(u, v)$ and the ratio between the largest and the smallest edge weight by W. The weight $w_G(F)$ of a set of edges $F \subseteq E$ is the sum of the individual edge weights. We will assume that all weights are polynomially bounded because there are standard reductions from the general case using minimum spanning trees (e.g. [55] Section 10.2., [56] Theorem 5.2). Also, these contraction schemes in the data structure setting introduces another layer of complexity akin to dynamic connectivity, which we believe is best studied separately.

A *dynamic algorithm* is a data structure for dynamically maintaining the result of a computation while the underlying input graph is updated periodically. We consider two types of updates: edge insertions and edge deletions. An *incremental* algorithm can handle only edge insertions, a *decremental* algorithm can handle only edge deletions, and a *fully dynamic* algorithm can handle both edge insertions and deletions. After every update in the graph, the dynamic algorithm is allowed to process the update to compute the new result. For the problem of maintaining a sparsifier, we want the algorithm to output the changes to the sparsifier (i.e., the edges to add to or remove from the sparsifier) after every update in the graph.

1.3.2 Running Times and Success Probabilities

The running time spent by the algorithm after every update is called *update time*. We distinguish between *amortized* and *worst-case* update time. A dynamic algorithm has amortized update time $T(m, n, W)$, if the total time spent after q updates in the graph is at most $qT(m, n, W)$. A dynamic algorithm has worst-case update time $T(m, n, W)$, if the total time spent after *each* update in the graph is at most $T(m, n, W)$. Here m refers to the maximum number of edges ever contained in the graph. All our algorithms are randomized.

The guarantees we report in this paper (quality and size of sparsifier, and update time) will hold *with high probability (w.h.p.)*, i.e. with probability at least $1 - 1/n^c$ for some arbitrarily chosen constant $c \geq 1$. These bounds are against an *oblivious adversary* who chooses its sequence of updates independently from the random choices made by the algorithm. Formally, the oblivious adversary chooses its sequence of updates before the algorithm starts. In particular, this means that the adversary is not allowed to see the current edges of the sparsifier. As our composition of routines involve $\mathrm{poly}(n)$ calls, we will assume the composability of these w.h.p. bounds.

Most of our update costs have the form $O(\log^{O(1)} n\epsilon^{-O(1)})$, where ϵ is the approximation error. We will often state these as $\mathrm{poly}(\log n, \epsilon^{-1})$ when the exponents exceed 3, and

explicitly otherwise.

1.3.3 Cuts and Laplacians

A *cut* $U \subseteq V$ of G is a subset of vertices whose removal makes G disconnected. We denote by $\partial_G(U)$ the edges crossing the cut U, i.e., the set of edges with one endpoint in U and one endpoint in $V \setminus U$. The weight of the cut U is $w_G(\partial_G(U))$. An *edge cut* $F \subseteq E$ of G is a a subset of edges whose removal makes G disconnected and the weight of the edge cut F is $w_G(F)$. For every pair of vertices u and v, the *local edge connectivity* $\lambda_G(u, v)$ is the weight of the minimum edge cut separating u and v. If G is unweighted, then $\lambda_G(u, v)$ amounts to the number of edges that have to be removed from G to make u and v disconnected.

Assuming some arbitrary order $v_1, \dots v_n$ on the vertices, the *Laplacian matrix* $\mathcal{L}_G$ of an undirected graph G is the $n \times n$ matrix that in row i and column j contains the negated weight $-w_G(v_i, v_j)$ of the edge (v_i, v_j) and in the i-th diagonal entry contains the weighted degree $\sum_{j=1}^{n} w_G(v_i, v_j)$ of vertex v_i. Note that Laplacian matrices are symmetric. The matrix $\mathcal{L}_e$ of an edge e of G is the $n \times n$ Laplacian matrix of the subgraph of G containing only the edge e. It is 0 everywhere except for a 2×2 submatrix.

For studying the spectral properties of G we treat the graph as a resistor network. For every edge $e \in E$ we define the *resistance* of e as $r_G(e) = 1/w_G(e)$. The *effective resistance* $R_G(e)$ of an edge $e = (v, u)$ is defined as the potential difference that has to be applied to u and v to drive one unit of current through the network. A closed form expression of the effective resistance is $R_G(e) = b_{u,v}^{\top} \mathcal{L}_G^{\dagger} b_{u,v}$, where $\mathcal{L}_G^{\dagger}$ is the Moore-Penrose pseudo-inverse of the Laplacian matrix of G and $b_{u,v}$ is the n-dimensional vector that is 1 at position u, -1 at position v, and 0 otherwise.

1.3.4 Graph Approximations

The goal of graph sparsification is to find sparse subgraphs, or similar small objects, that approximately preserve certain metrics of the graph. We first define spectral sparsifiers

where we require that Laplacian quadratic form of the graph is preserved approximately. Spectral sparsifiers play a pivotal role in fast algorithms for solving Laplacian systems, a special case of linear systems.

Definition 1.3.1. A $(1 \pm \epsilon)$-*spectral sparsifier* H of a graph G is a subgraph of G with weights w_H such that for every vector $x \in \mathbb{R}^n$

$$(1-\epsilon)x^\top \mathcal{L}_H x \leq x^\top \mathcal{L}_G x \leq (1+\epsilon)x^\top \mathcal{L}_H x\,.$$

Using the Loewner ordering on matrices this condition can also be written as $(1-\epsilon)\mathcal{L}_H \preceq \mathcal{L}_G \preceq (1+\epsilon)\mathcal{L}_H$. An $n \times n$ matrix $\mathcal{A}$ is *positive semi-definite*, written as $\mathcal{A} \succeq 0$, if $x^\top \mathcal{A} x \geq 0$ for all $x \in \mathbb{R}^n$. For two $n \times n$ matrices $\mathcal{A}$ and $\mathcal{B}$ we write $\mathcal{A} \succeq \mathcal{B}$ as an abbreviation for $\mathcal{A} - \mathcal{B} \succeq 0$.

Note that $x^\top \mathcal{L}_G x = \sum_{(u,v)\in E} w(u,v)(x(u)-x(v))^2$ where the vector x is treated as a function on the vertices and $x(v)$ is the value of x for vertex v. A special case of such a function on the vertices is given by the binary indicator vector x_U associated with a cut U, where $x_U(v) = 1$ is $v \in U$ and 0 otherwise. If limited to such indicator vectors, the sparsifier approximately preserves the value of every cut.

Definition 1.3.2. A $(1 \pm \epsilon)$-*cut sparsifier* H of a graph G is a subgraph of G with weights w_H such that for every subset $U \subseteq V$

$$(1-\epsilon)w_H(\partial_H(U)) \leq w_G(\partial_G(U)) \leq (1+\epsilon)w_H(\partial_H(U))\,.$$

1.3.5 Sampling Schemes for Constructing Sparsifiers

Most efficient constructions of sparsifiers are randomized, partly because when G is the complete graph, the resulting sparsifier needs to be an expander. These randomized schemes rely on importance sampling, which for each edge:

1. Keeps it with probability p_e,

2. If the edge is kept, its weight is rescaled to $\frac{w_e}{p_e}$.

A crucial property of this process is that the edge's expectation is preserved. As both cut and spectral sparsifiers can be viewed as preserving sums over linear combinations of edge weights, each of these terms have correct expectation. The concentration of such processes can then be bounded using either matrix concentration bounds in the spectral case [57, 55], or a variety of combinatorial arguments [33].

Our algorithms in this paper will use an even simpler version of this importance sampling scheme: all of our p_e's will be set to either 1 or $1/2$. This scheme has a direct combinatorial interpretation:

1. Keep some of the edges.

2. Take a random half of the other edges, and double the weights of the edges kept.

Note that composing such a routine $O(\log n)$ times gives a sparsifier, as long as the part we keep is small. So the main issue is to figure out how to get a small part to keep.

1.3.6 Spanning Trees and Spanners

A *spanning forest* F of G is a forest (i.e., acyclic graph) on a subset of the edges of G such that every pair of vertices that is connected in G is also connected in F. A minimum/maximum spanning forest is a spanning forest of minimum/maximum total weight.

For every pair of vertices u and v we denote by $d_G(u, v)$ the distance between u and v (i.e., the length of the shortest path connecting u and v) in G with respect to the resistances. The graph sparsification concept also exists with respect to distances in the graph. Such sparse subgraphs that preserves distances approximately are called spanners.

Definition 1.3.3. A *spanner of stretch* α, or short α*-spanner*, (where $\alpha \geq 1$) of an undirected (possibly weighted) graph G is a subgraph H of G such that, for every pair of vertices u and v, $d_H(u, v) \leq \alpha d_G(u, v)$.

1.4 Overview and Related Work

1.4.1 Dynamic Spectral Sparsifier

We first develop a fully dynamic algorithm for maintaining a spectral sparsifier of a graph with polylogarithmic amortized update time.

Related Work. Spectral sparsifiers play important roles in fast numerical algorithms. Spielman and Teng were the first to study these objects [34]. Their algorithm constructs a $(1 \pm \epsilon)$-spectral sparsifier of size $O(n \cdot \text{poly}(\log n, \epsilon^{-1}))$ in nearly linear time. This result has seen several improvements in recent years [55, 58, 59, 51]. The state of the art in the sequential model is an algorithm by Lee and Sun [52] that computes a $(1 \pm \epsilon)$-spectral sparsifier of size $O(n\epsilon^{-2})$ in nearly linear time. Most closely related to the data structural question are streaming routines, both in one pass incremental [48], and turnstile [60, 61, 50].

A survey of spectral sparsifier constructions is given in [35]. Many of these methods rely on solving linear systems built on the graph, for which there approaches with a combinatorial flavor using low-stretch spanning trees [62, 63] and purely numerical solvers relying on sparsifiers [39] or recursive constructions [41]. The crux of these algorithms is then a simple sampling procedure of each edge independently kept with probability proportional to its respective effective resistance. The notion of effective resistance and sparsification sampling based upon this metric will be discussed more in depth in Section 2 (see Section 2.3.2 for a definition of effective resistance) where we slightly modify this sampling procedure to maintain further properties of the graph. It is important to note that the metric of effective resistance is a global quantity of the graph and is difficult to dynamically maintain efficiently. As such, while sampling by effective resistance score may give the smallest sparsifier in the static case, it is difficult to extend to the dynamic setting. We instead build on the spectral sparsifier obtained by a simple, combinatorial construction of Koutis [43], which initially was geared towards parallel and distributed implementations and will be further explained in

later sections. The key distinction will be that the sampling procedure for most edges is not affected by the insertion or deletion of an edge allowing for more efficient dynamic updates.

Sparsification Framework. In our framework we determine 'sampleable' edges by using spanners to compute a set of edges of bounded effective resistance. From these edges we then sample by coin flipping to obtain a (moderately sparser) spectral sparsifier in which the number of edges has been reduced by a constant fraction. This step can then be iterated a small number of times in order to compute the final sparsifier.

Concretely, we define a t-bundle spanner $B = T_1 \cup \cdots \cup T_t$ (for a suitable, polylogarithmic, value of t) as a sequence of spanners $T_1, \ldots, T_t$ where the edges of each spanner are removed from the graph before computing the next spanner, i.e., T_1 is a spanner of G, T_2 is a spanner of $G \setminus T_1$, etc; here each spanner has stretch $O(\log n)$. We then sample each non-bundle edge in $G \setminus B$ with some constant probability p and scale the edge weights of the sampled edges proportionally. The t-bundle spanner serves as a certificate for small resistance of the non-bundle edges in $G \setminus B$ as it guarantees the presence of t disjoint paths of length at most the stretch of the spanner. Using this property one can apply matrix concentration bounds [57] to show the t-bundle together with the sampled edges is a moderately sparse spectral sparsifier. We repeat this process of 'peeling off' a t-bundle from the graph and sampling from the remaining edges until the graph is sparse enough (which happens after a logarithmic number of iterations). Our final sparsifier consists of all t-bundles together with the sampled edges of the last stage.

Towards a Dynamic Algorithm. To implement the spectral sparsification algorithm in the dynamic setting we need to dynamically maintain a t-bundle spanner. Our approach to this problem is to run t different instances of a dynamic spanner algorithm, in order to separately maintain a spanner T_i for each graph $G_i = G \setminus \bigcup_{j=1}^{i-1} T_j$, for $1 \leq i \leq t$.

Baswana, Khurana, and Sarkar [29] gave a fully dynamic algorithm for maintaining

a spanner of stretch $O(\log n)$ and size $O(n \log^2 n)$ with polylogarithmic update time.[1] A natural first idea would be to use this algorithm in a black-box fashion in order to separately maintain each spanner of a t-bundle. However, we do not know how to do this because of the following obstacle. A single update in G might lead to several changes of edges in the spanner T_1, an average of $\Omega(\log n)$ according to the amortized upper bound. This means that the next instance of the fully dynamic spanner algorithm which is used for maintaining T_2, not only has to deal with the deletion in G but also the artificially created updates in $G_2 = G \setminus T_1$. This of course propagates to more updates in all graphs G_i. Observe also that any given update in G_t caused by an update in G, can be requested *repeatedly*, as a result of subsequent updates in G. Without further guarantees, it seems that with this approach we can only hope for an upper bound of $O(\log^{t-1} n)$ (on average) on the number of changes to be processed for updating G_t after a single update in G. That is too high because the sparsification algorithm requires us to take $t = \Omega(\log n)$. Our solution to this problem lies in a substantial modification of the dynamic spanner algorithm in [29] outlined below.

Dynamic Spanners with Monotonicity. The spanner algorithm of [29] is at its core a decremental algorithm (i.e., allowing only edge deletions in G), which is subsequently leveraged into a fully dynamic algorithm by a black-box reduction. We follow the same approach by first designing a decremental algorithm for maintaining a t-bundle spanner. This is achieved by modifying the decremental spanner algorithm so that, in addition to its original guarantees, it has the following **monotonicity** property:

Every time an edge is added to the spanner T, it stays in T until it is deleted from G.

Recall that we initially want to maintain a t-bundle spanner $T_1, \ldots, T_t$ under edge deletions only. In general, whenever an edge is added to T_1, it will cause its deletion from the graph $G \setminus T_1$ for which the spanner T_2 is maintained. Similarly, removing an edge from T_1

[1] More precisely, they gave two fully dynamic algorithms for maintaing a $(2k-1)$-spanner for any integer $k \geq 2$: The first algorithm guarantees a spanner of expected size $O(kn^{1+1/k} \log n)$ and has expected amortized update time $O(k^2 \log^2 n)$ and the second algorithm guarantees a spanner of expected size $O(k^8 n^{1+1/k} \log^2 n)$ and has expected amortized update time $O(7^{k/2})$.

causes its insertion into $G \setminus T_1$, *unless* the edge is deleted from G. This is precisely what the monotonicity property guarantees: that an edge will not be removed from T_1 unless deleted from G. The consequence is that no edge insertion can occur for $G_2 = G \setminus T_1$. Inductively, no edge is ever inserted into G_i, for each i. Therefore the algorithm for maintaining the spanner T_i only has to deal with edge deletions from the graph G_i, thus it becomes possible to run a different instance of the same decremental spanner algorithm for each G_i. A single deletion from G can still generate many updates in the bundle. But for each i, the instance of the dynamic spanner algorithm working on G_i can only delete each edge *once*. Furthermore, we only run a small number t of instances. So the total number of updates remains bounded, allowing us to claim the upper bound on the amortized update time.

In addition to the modification of the dynamic spanner algorithm, we have also deviated from Koutis' original scheme [43] in that we explicitly 'peel off' each iteration's bundle from the graph. In this way we avoid that the t-bundles from different iterations share any edges, which seems hard to handle in the decremental setting we ultimately want to restrict ourselves to.

The modified spanner algorithm now allows us to maintain t-bundles in polylogarithmic update time, which is the main building block of the sparsifier algorithm. The remaining parts of the algorithm, like sampling of the non-bundle edges by coin-flipping, can now be carried out in the straightforward way in polylogarithmic amortized update time. At any time, our modified spanner algorithm can work in a purely decremental setting. As mentioned above, the fully dynamic sparsifier algorithm is then obtained by a reduction from the decremental sparsifier algorithm.

1.4.2 Dynamic Cut Sparsifier

We then give dynamic algorithms for maintaining a $(1 \pm \epsilon)$-cut sparsifier. We obtain a fully dynamic algorithm with polylogarithmic worst-case update time by leveraging a recent worst-case update time algorithm for dynamically maintaining a spanning tree of a

graph [24]. As mentioned above, spectral sparsifiers are more general than cut sparsifiers. The big advantage of studying cut sparsification as a separate problem is that we can achieve polylogarithmic *worst-case* update time, where the update time guarantee holds for each individual update and is *not* amortized over a sequence of updates.

Related Work. In the static setting, Benczúr and Karger [33] developed an algorithm for computing a $(1 \pm \epsilon)$-cut sparsifier of size $O(n \cdot \text{poly}(\log n, \epsilon^{-1}))$ in nearly linear time. Their approach is to first compute a value called *strength* for each edge and then sampling each edge with probability proportional to its strength. Their proof uses a cut-counting argument that shows that the majority of cuts are large, and therefore less likely to deviate from their expectation. A union bound over these (highly skewed) probabilities then gives the overall w.h.p. success bound. This approach was refined by Fung et al. [64] who show that a cut sparsifier can also be obtained by sampling each edge with probability inversely proportional to its (approximate) local edge connectivity, giving slightly better guarantees on the sparsifier. The work of Kapron, King, and Mountjoy [24] contains a fully dynamic approximate "cut oracle" with worst-case update time $O(\log^2 n)$. Given a set $U \subseteq V$ as the input of a query, it returns a 2-approximation to the number of edges in $U \times V \setminus U$ in time $O(|U| \log^2 n)$. The cut sparsifier question has also been studied in the (dynamic) streaming model [65, 66, 67].

Our Framework. The algorithm is based on the observation that the spectral sparsification scheme outlined above in Section 1.4.1. becomes a cut sparsification algorithm if we simply replace spanners by maximum weight spanning trees (MSTs). This is inspired by sampling according to edge connectivities; the role of the MSTs is to certify lower bounds on the edge connectivities. We observe that the framework does not require us to use exact MSTs. For our t-bundles we can use a relaxed, approximate concept that we call α-MST that. Roughly speaking, an α-MST guarantees a 'stretch' of α in the infinity norm and, as long as it is sparse, does not necessarily have to be a tree.

Similarly to before, we define a t-bundle α-MST B as the union of a sequence of α-MSTs $T_1, \ldots T_t$ where the edges of each tree are removed from the graph before computing the next α-MST. The role of α-MST is to certify uniform lower bounds on the connectivity of edges; these bounds are sufficiently large to allow uniform sampling with a fixed probability.

This process of peeling and sampling is repeated sufficiently often and our cut sparsifier then is the union of all the t-bundle α-MSTs and the non-bundle edges remaining after taking out the last bundle. Thus, the cut sparsifier consists of a polylogarithmic number of α-MSTs and a few (polylogarithmic) additional edges. This means that for α-MSTs based on spanning trees, our cut sparsifiers are not only sparse, but also have polylogarithmic *arboricity*, which is the minimum number of forests into which a graph can be partitioned.

Simple Fully Dynamic Algorithm. Our approach immediately yields a fully dynamic algorithm by using a fully dynamic algorithm for maintaining a spanning forest. Here we basically have two choices. Either we use the randomized algorithm of Kapron, King, and Mountjoy [24] with polylogarithmic worst-case update time. Or we use the deterministic algorithm of Holm, de Lichtenberg, and Thorup [23] with polylogarithmic amortized update time. The latter algorithm is slightly faster, at the cost of providing only amortized update-time guarantees. A t-bundle 2-MST can be maintained fully dynamically by running, for each of the $\log W$ weight classes of the graph, t instances of the dynamic spanning tree algorithm in a 'chain'.

An important observation about the spanning forest algorithm is that with every update in the graph, at most one edge is changed in the spanning forest: If for example an edge is deleted from the spanning forest, it is replaced by another edge, but no other changes are added to the tree. Therefore a single update in G can only cause one update for each graph $G_i = G \setminus \bigcup_{j=1}^{i-1} T_j$ and T_i. This means that each instance of the spanning forest algorithm creates at most one 'artificial' update that the next instance has to deal with. In this way, each dynamic spanning forest instance used for the t-bundle has polylogarithmic update

time. As $t = \text{polylog}\, n$, the update time for maintaining a t-bundle is also polylogarithmic. The remaining steps of the algorithm can be carried out dynamically in the straightforward way and overall give us polylogarithmic worst-case or amortized update time.

A technical detail of our algorithm is that the high-probability correctness achieved by the Chernoff bounds only holds for a polynomial number of updates in the graph. We thus have to restart the algorithm periodically. This is trivial when we are shooting for an amortized update time. For a worst-case guarantee we can neither completely restart the algorithm nor change all edges of the sparsifier in one time step. We therefore keep two instances of our algorithm that maintain two sparsifiers of two alternately growing and shrinking subgraphs that at any time partition the graph. This allows us to take a blend of these two subgraph sparsifiers as our end result and take turns in periodically restarting the two instances of the algorithm.

1.4.3 $(1-\epsilon)$-Approximate Undirected Bipartite Flow

We then study ways of utilizing our sparsifier constructions to give routines with truly sublinear update times. The problem that we work with will be maintaining an approximate maximum flow problem on a bipartite graph $G_{A,B} = (A, B, E)$ with demand -1 and 1 on each vertex in A and B, respectively. All edges are unit weight and we dynamically insert and delete edges. The maximum flow minimum cut theorem states that the objective here equals to the minimum $s-t$ cut or maximum $s-t$ flow in G, which will be $G_{A,B}$ where we add vertices s and t, and connect each vertex in A to s and each vertex in B to t. The only dynamic changes in this graph will be in edges between A and B. As our algorithms builds upon cut sparsifiers, and flow sparsifiers [38] are more involved, we will focus on only finding cuts.

This problem is motivated by the dynamic approximate maximum matching problem, which differs in that the edges are directed, and oriented from A to B. This problem has received much attention recently [25, 27, 26, 68, 54, 69], and led to the key definition of low

arboricity graphs [26, 54]. On the other hand, bipartite graphs are known to be difficult to sparsify: the directed reachability matrix from A to B can encode $\Theta(n^2)$ bits of information. As a result, we study the undirected variant of this problem instead, with the hope that this framework can motivate other definitions of sparsification suitable for wider classes of graphs.

Another related line of work are fully dynamic algorithm for maintaining the global minimum cut [70, 71] with update time $O(\sqrt{n}\operatorname{polylog} n)$. As there are significant differences between approximating global minimum cuts and st-minimum cuts in the static setting [72], we believe that there are some challenges to adapting these techniques for this problem. The data structure by Thorup [70] can either maintain global edge connectivity up to $\operatorname{polylog} n$ exactly or, with high probability, arbitrary global edge connectivity with an approximation of $1 + o(1)$. The algorithms also maintain concrete (approximate) minimum cuts, where in the latter algorithm the update time increases to $O(\sqrt{m}\operatorname{polylog} n)$ (and cut edges can be listed in time $O(\log n)$ per edge). Thorup's result was preceded by a randomized algorithm with worse approximation ratio for the global edge connectivity by Thorup and Karger [71] with update time $O(\sqrt{n}\operatorname{polylog} n)$.

At the start of Section 1.7 we will show that the problem we have formulated above is in fact different from matching. On the other hand, our incorporation of sparsifiers for maintaining solutions to this problem relies on several properties that hold in a variety of other settings:

1. The static version can be efficiently approximated.
2. The objective can be approximated via graph sparsifiers.
3. A small answer (for which the algorithm's current approximation may quickly become sub-optimal) means the graph also has a small vertex cover.
4. The objective does not change much per each edge update.

As with algorithms for maintaining high quality matchings [68, 54], our approach aims to get a small amortized cost by keeping the same minimum $s - t$ cut for many consecutive dynamic steps. Specifically, if we have a minimum $s - t$ cut of size $(2 + \frac{\epsilon}{2})OPT$, then we know this cut will remain $(2 + \epsilon)$ approximately optimal for $\frac{\epsilon}{2}OPT$ dynamic steps. This allows us to only compute a new minimum $s - t$ cut every $\frac{\epsilon}{2}OPT$ dynamic steps.

As checking for no edges would be an easy boundary case, we will assume throughout all the analysis that $OPT > 0$. To obtain an amortized $O(\text{poly}(\log n, \epsilon^{-1}))$ update cost, it suffices for this computation to take $O(OPT \cdot \text{poly}(\log n, \epsilon^{-1}))$ time. In other words, we need to solve approximate maximum flow on a graph of size $O(OPT \cdot \text{poly}(\log n, \epsilon^{-1}))$. Here we incorporate sparsifiers using the other crucial property used in matching data structures [25, 68, 54]: if OPT is small, G also has a small vertex cover.

Lemma 1.4.1. *The minimum vertex cover in G has size at most $OPT + 2$ where OPT is the size of the minimum $s - t$ cut in G.*

We utilize the low arboricity of our sparsifiers to find a small vertex cover with the additional property that all non-cover vertices have small degree. We will denote this (much) smaller set of vertices as VC. In a manner similar to eliminating vertices in numerical algorithms [41], the graph can be reduced to only edges on VC at the cost of a $(2 + \epsilon)$-approximation. Maintaining a sparsifier of this routine again leads to an overall routine that maintains a $(2 + \epsilon)$-approximation in $\text{polylog}\, n$ time per update, which we show in Section 1.7.

Sparsifying vertices instead of edges inherently implies that an approximation of all cut values cannot be maintained. Instead, the sparsifier, which will be referred to as a *terminal-cut-sparsifier*, maintains an approximation of all minimum cuts between any two terminal vertices, where the vertex cover is the terminal vertex set for our purposes. More specifically, given a minimum cut between two terminal vertices on the sparsified graph, by adding each independent vertex from the original graph to the cut set it is more connected to, an approximate minimum cut on the original graph is achieved. This concept

of *terminal-cut-sparsifier* will be equivalent to that in [2], and will be given formal treatment in Section 1.9.

The large approximation ratio motivated us to reexamine the sparsification routines, namely the one of reducing the graph to one whose size is proportional to $|VC|$. This is directly related to the terminal cut sparsifiers studied in [2, 73]. However, for an update time of $\text{poly}(\log n, \epsilon^{-1})$, it is crucial for the vertex sparsifier to have size $O(|VC| \text{ poly}(\log n, \epsilon^{-1}))$. As a result, instead of doing a direct union bound over all $2^{|VC|}$ cuts to get a size of $\text{poly}(|VC|)$ as in [2], we need to invoke cut counting as with cut sparsifier constructions. This necessitates the use of objects similar to t-bundles to identify edges with small connectivity. This leads to a sampling process motivated by the $(2+\epsilon)$-approximate routine, but works on vertices instead of edges.

By relating the processes, we are able to absorb the factor 2 error into the sparsifier size. In Section 1.8, we formalize this process, as well as its guarantees on graphs with bounded weights. Here a major technical challenge compared to analyses of cut sparsifiers [64] is that the natural scheme of bucketing by edge weights is difficult to analyze because a sampled vertex could have non-zero degree in multiple buckets. We work around this issue via a pre-processing scheme on G that creates an approximation so that all vertices outside of VC have degree $\text{polylog}\, n$. This scheme is motivated in part by the weighted expanders constructions from [41]. Bucketing after this processing step ensures that each vertex belongs to a unique bucket. In terms of a static sparsifier on terminals, the result that is most comparable to results from previous works is:

Corollary 1.4.2. *Given any graph $G = (V, E)$, and a vertex cover VC of G, where $X = V \setminus VC$, with error ϵ, we can build an ϵ-approximate terminal-cut-sparsifier H with $O(|VC| \text{ poly}(\log n, \epsilon^{-1}))$ vertices in $O(m \cdot \text{poly}(\log n, \epsilon^{-1}))$ work.*

Turning this into a dynamic routine leads to the result described in Theorem 1.2.3: a $(1+\epsilon)$-approximate solution that can be maintained in time $\text{polylog}(n)$ per update. It is important to note that Theorem 1.2.2 plays an integral role in extending Corollary 1.4.2 to a

dynamic routine, particularly the low arboricity property that allows us to maintain a small vertex cover such that all non-cover vertices have low degree. These algorithmic extensions, as well as their incorporation into data structures are discussed in Section 1.9.

1.4.4 Discussion

Graph Sparsification. We use a sparsification framework in which we 'peel off' bundles of sparse subgraphs to determine 'sampleable' edges, from which we then sample by coin flipping. This leads to combinatorial and surprisingly straightforward algorithms for maintaining graph sparsifiers. Additionally, this gives us low-arboricity sparsifiers; a property that we exploit for our main application.

Although spectral sparsification is more general than cut sparsification. Our treatment of cut sparsification has two motivations. First, we can obtain stronger running time guarantees. Second, our sparsifier for the $(1-\epsilon)$-approximate maximum flow algorithm on bipartite graphs hinges upon improved routines for vertex sparsification, a concept which leads to different objects in the spectral setting.

Dynamic Graph Algorithms. In our sparsification framework we sequentially remove bundles of sparse subgraphs to determine 'sampleable' edges. This leads to 'chains' of dynamic algorithms where the output performed by one algorithm might result in updates to the input of the next algorithm. This motivates a more fine-grained view on of dynamic algorithms with the goal of obtaining strong bounds on the number of changes to the output.

Future Work. The problem whether spectral sparsifiers can be maintained with polylogarithmic *worst-case* update time remains open. Our construction goes via spanners and therefore a natural question is whether spanners can be maintained with worst-case update time. Maybe there are also other more direct ways of maintaining the sparsifier. A more general question is whether we can find more dynamic algorithms for numerical problems.

Our dynamic algorithms cannot avoid storing the original graph, which is undesirable

in terms of space consumption. Can we get space-efficient dynamic algorithms without sacrificing fast update time?

The sparsification framework for peeling off subgraphs and uniformly sampling from the remaining edges is very general. Are there other sparse subgraphs we could start with in the peeling process? Which properties do the sparsifiers obtained in this way have? In particular, it would be interesting to see whether our techniques can be generalized to flow sparsifiers [38, 2].

The combination of sparsifiers with density-sensitive approaches for dynamic graph data structures [26, 54] provides an approach for obtaining $\mathrm{poly}(\log, \epsilon^{-1})$ update times. We believe this approach can be generalized to other graph cut problems. In particular, the flow networks solved for balanced cuts and graph partitioning are also bipartite and undirected, and therefore natural directions for future work.

1.5 Dynamic Spectral Sparsifier

In this section we give an algorithm for maintaining a spectral sparsifier under edge deletions and insertions with polylogarithmic amortized update time. The main result of this section is as follows.

Theorem 1.5.1. *There exists a fully dynamic randomized algorithm with polylogarithmic update time for maintaining a $(1 \pm \epsilon)$-spectral sparsifier H of a graph G, with probability at least $1 - 1/n^c$ for any $0 < \epsilon \leq 1$ and $c \geq 1$. Specifically, the amortized update time of the algorithm is*

$$O(c\epsilon^{-2} \log^3 \rho \log^6 n)$$

and the size of H is

$$O(cn\epsilon^{-2} \log^3 \rho \log^5 n \log W + m\rho^{-1}),$$

where $1 \leq \rho \leq m$ is a parameter of choice. Here, W is the ratio between the largest and the smallest edge weight in G. The ratio between the largest and the smallest edge weight in

H is at most $O(nW)$.

After giving an overview of our algorithm, we first explain our spectral sparsification scheme in a static setting and prove its properties. Subsequently, we show how we can dynamically maintain the edges of such a sparsifier by making this scheme dynamic.

1.5.1 Algorithm Overview

Sparsification Framework. In our framework we determine 'sampleable' edges by using spanners to compute a set of edges of bounded effective resistance. From these edges we then sample by coin flipping to obtain a (moderately sparser) spectral sparsifier in which the number of edges has been reduced by a constant fraction. This step can then be iterated a small number of times in order to compute the final sparsifier.

Concretely, we define a t-bundle spanner $B = T_1 \cup \cdots \cup T_t$ (for a suitable, polylogarithmic, value of t) as a sequence of spanners $T_1, \ldots, T_t$ where the edges of each spanner are removed from the graph before computing the next spanner, i.e., T_1 is a spanner of G, T_2 is a spanner of $G \setminus T_1$, etc; here each spanner has stretch $O(\log n)$. We then sample each non-bundle edge in $G \setminus B$ with some constant probability p and scale the edge weights of the sampled edges proportionally. The t-bundle spanner serves as a certificate for small resistance of the non-bundle edges in $G \setminus B$ as it guarantees the presence of t disjoint paths of length at most the stretch of the spanner. Using this property one can apply matrix concentration bounds to show the t-bundle together with the sampled edges is a moderately sparse spectral sparsifier. We repeat this process of 'peeling off' a t-bundle from the graph and sampling from the remaining edges until the graph is sparse enough (which happens after a logarithmic number of iterations). Our final sparsifier consists of all t-bundles together with the sampled edges of the last stage.

Towards a Dynamic Algorithm. To implement the spectral sparsification algorithm in the dynamic setting we need to dynamically maintain a t-bundle spanner. Our approach

to this problem is to run t different instances of a dynamic spanner algorithm, in order to separately maintain a spanner T_i for each graph $G_i = G \setminus \bigcup_{j=1}^{i-1} T_j$, for $1 \leq i \leq t$.

Baswana, Khurana, and Sarkar [29] gave a fully dynamic algorithm for maintaining a spanner of stretch $O(\log n)$ and size $O(n \log^2 n)$ with polylogarithmic update time.[2] A natural first idea would be to use this algorithm in a black-box fashion in order to separately maintain each spanner of a t-bundle. However, we do not know how to do this because of the following obstacle. A single update in G might lead to several changes of edges in the spanner T_1, an average of $\Omega(\log n)$ according to the amortized upper bound. This means that the next instance of the fully dynamic spanner algorithm which is used for maintaining T_2, not only has to deal with the deletion in G but also the artificially created updates in $G_2 = G \setminus T_1$. This of course propagates to more updates in all graphs G_i. Observe also that any given update in G_t caused by an update in G, can be requested *repeatedly*, as a result of subsequent updates in G. Without further guarantees, it seems that with this approach we can only hope for an upper bound of $O(\log^{t-1} n)$ (on average) on the number of changes to be processed for updating G_t after a single update in G. That is too high because the sparsification algorithm requires us to take $t = \Omega(\log n)$. Our solution to this problem lies in a substantial modification of the dynamic spanner algorithm in [29] outlined below.

Dynamic Spanners with Monotonicity. The spanner algorithm of [29] is at its core a decremental algorithm (i.e., allowing only edge deletions in G), which is subsequently leveraged into a fully dynamic algorithm by a black-box reduction. We follow the same approach by first designing a decremental algorithm for maintaining a t-bundle spanner. This is achieved by modifying the decremental spanner algorithm so so that, additional to its original guarantees, it has the following **monotonicity** property:

Every time an edge is added to the spanner T, it stays in T until it is deleted from G.

[2]More precisely, they gave two fully dynamic algorithms for maintaing a $(2k-1)$-spanner for any integer $k \geq 2$: The first algorithm guarantees a spanner of expected size $O(kn^{1+1/k} \log n)$ and has expected amortized update time $O(k^2 \log^2 n)$ and the second algorithm guarantees a spanner of expected size $O(k^8 n^{1+1/k} \log^2 n)$ and has expected amortized update time $O(7^{k/2})$.

Recall that we initially want to maintain a t-bundle spanner $T_1, \ldots, T_t$ under edge deletions only. In general, whenever an edge is added to T_1, it will cause its deletion from the graph $G \setminus T_1$ for which the spanner T_2 is maintained. Similarly, removing an edge from T_1 causes its insertion into $G \setminus T_1$, *unless* the edge is deleted from G. This is precisely what the monotonicity property guarantees: that an edge will not be removed from T_1 unless deleted from G. The consequence is that no edge insertion can occur for $G_2 = G \setminus T_1$. Inductively, no edge is ever inserted into G_i, for each i. Therefore the algorithm for maintaining the spanner T_i only has to deal with edge deletions from the graph G_i, thus it becomes possible to run a different instance of the same decremental spanner algorithm for each G_i. A single deletion from G can still generate many updates in the bundle. But for each i the instance of the dynamic spanner algorithm working on G_i can only delete each edge *once*. Furthermore, we only run a small number t of instances. So the total number of updates remains bounded, allowing us to claim the upper bound on the amortized update time.

In addition to the modification of the dynamic spanner algorithm, we have also deviated from Koutis' original scheme [43] in that we explicitly 'peel off' each iteration's bundle from the graph. In this way we avoid that the t-bundles from different iterations share any edges, which seems hard to handle in the decremental setting we ultimately want to restrict ourselves to.

The modified spanner algorithm now allows us to maintain t-bundles in polylogarithmic update time, which is the main building block of the sparsifier algorithm. The remaining parts of the algorithm, like sampling of the non-bundle edges by coin-flipping, can now be carried out in the straightforward way in polylogarithmic amortized update time. At any time, our modified spanner algorithm can work in a purely decremental setting. As mentioned above, the fully dynamic sparsifier algorithm is then obtained by a reduction from the decremental sparsifier algorithm.

1.5.2 Spectral Sparsification

As outlined above, iteratively 'peels off' bundles of spanners from the graph.

Definition 1.5.2. A *t-bundle α-spanner* (where $t \geq 1$, $\alpha \geq 1$) of an undirected graph G is the union $T = \bigcup_{i=1}^{k} T_i$ of a sequence of graphs $T_1, \ldots, T_k$ such that, for every $1 \leq i \leq k$, T_i is an α-spanner of $G \setminus \bigcup_{j=1}^{i-1} T_j$.

The algorithm for spectral sparsification is presented in Figures 1.1 and 1.2. Algorithm LIGHT-SPECTRAL-SPARSIFY computes a moderately sparser $(1 \pm \epsilon)$-spectral sparsifier. Algorithm SPECTRAL-SPARSIFY takes a parameter ρ and computes the sparsifier in $k = \lceil \log \rho \rceil$ iterations of LIGHT-SPECTRAL-SPARSIFY.

We will now prove the properties of these algorithms. We first need the following lemma that shows how t-bundle spanners can be used to bound effective resistances. We highlight the main intuition of this crucial observation in our proof sketch.

Lemma 1.5.3 ([43])**.** *Let G be a graph and B be a t-bundle α-spanner of G. For every edge e of $G \setminus B$, we have*

$$w_G(e) \cdot R_G(e) \leq \frac{\alpha}{t}$$

which implies that

$$w_G(e) \cdot \mathcal{L}_e \preceq \frac{\alpha}{t} \cdot \mathcal{L}_G$$

where $\mathcal{L}_e$ is the $n \times n$ Laplacian of the unweighted edge e.

Sketch. Fix some edge $e = (u, v)$ of $G \setminus B$ and let $T_1, \ldots T_t$ denote the (pairwise disjoint) α-spanners contained in B. For every $1 \leq i \leq t$, let π_i denote the shortest path from u to v in T_i. The length of the path π in T_i exceeds the distance from u to v in $G \setminus \bigcup_{j=1}^{i-1} T_j$ by at most a factor of α (property of the spanner T_i). Since e is contained in $G \setminus B$, the latter distance is at most the resistance of the edge e as we have defined distances as the length of shortest paths with respect to the resistances of the edges.

LIGHT-SPECTRAL-SPARSIFY (G, ϵ)

1. $t \leftarrow \lceil 12(c+1)\alpha\epsilon^{-2}\ln n \rceil$ for some absolute constant c.
2. let $B = \bigcup_{j=1}^{t} T_j$ be a t-bundle α-spanner of G
3. $H := B$
4. **for each** edge $e \in G \setminus B$
   (a) with probability $1/4$: add e to H with $w_H(e) \leftarrow 4w_G(e)$
5. **return** (H, B)

Figure 1.1: LIGHT-SPECTRAL-SPARSIFY (G, c, ϵ). We give a dynamic implementation of this algorithm in Section 1.5.4. In particular we dynamically maintain the t-bundle α-spanner B which results in a dynamically changing graph $G \setminus B$.

SPECTRAL-SPARSIFY (G, c, ϵ)

1. $k \leftarrow \lceil \log \rho \rceil$
2. $G_0 \leftarrow G$
3. $B_0 \leftarrow (V, \emptyset)$
4. **for** $i = 1$ **to** k
   (a) $(H_i, B_i) \leftarrow$ LIGHT-SPECTRAL-SPARSIFY$(G_{i-1}, c, \epsilon/(2k))$
   (b) $G_i \leftarrow H_i \setminus B_i$
   (c) **if** G_i has less than $(c+1)\ln n$ edges **then break** (* break loop *)
5. $H \leftarrow \bigcup_{1 \le j \le i} B_j \cup G_i$
6. **return** $(H, \{B_j\}_{j=1}^{i}, G_i)$

Figure 1.2: SPECTRAL-SPARSIFY (G, c, ϵ). We give a dynamic implementation of this algorithm in Section 1.5.4. In particular we dynamically maintain each H_i and B_i as the result of a dynamic implementation of LIGHT-SPECTRAL-SPARSIFY which results in dynamically changing graphs G_i.

Consider each path π_i as a subgraph of G and let Π be the subgraph consisting of all paths π_i. Observe that Π consists of a parallel composition of paths, which in turn consists of a serial composition of edges, the we can view as resistors. We can now apply the well-known rules for serial and parallel composition for computing effective resistances and get the desired bounds. □

Our second tool in the analysis the following variant [74] of a matrix concentration inequality by Tropp [57].

Theorem 1.5.4. *Let $Y_1, \ldots, Y_k$ be independent positive semi-definite matrices of size $n \times n$. Let $Y = \sum_{i=1}^{k} Y_i$ and $Z = \mathbb{E}[Y]$. Suppose $Y_i \preceq RZ$, where R is a scalar, for every $1 \leq i \leq k$. Then for all $\epsilon \in [0, 1]$*

$$\mathbb{P}\left[\sum_{i=1}^{k} Y_i \preceq (1-\epsilon)Z\right] \leq n \cdot \exp(-\epsilon^2/2R)$$
$$\mathbb{P}\left[\sum_{i=1}^{k} Y_i \succeq (1+\epsilon)Z\right] \leq n \cdot \exp(-\epsilon^2/3R)$$

Given these facts we can now prove the following Lemma which is a slight generalization of a Lemma in [43]. As the proof is quite standard we have moved it to Appendix 1.10 (together with the proofs of the subsequent two lemmas). For applying the lemma in our dynamic algorithm it is crucial that the input graph (which might be generated by another randomized algorithm) is independent of the random choices of algorithm LIGHT-SPECTRAL-SPARSIFY.

Lemma 1.5.5. *The output H of* LIGHT-SPECTRAL-SPARSIFY *is a $(1\pm\epsilon)$-spectral sparsifier with probability at least $1 - n^{-(c+1)}$ for any input graph G that is independent of the random choices of the algorithm.*

By iteratively applying the sparsification of LIGHT-SPECTRAL-SPARSIFY as done in SPECTRAL-SPARSIFY we obtain sparser and sparser cut sparsifiers.

Lemma 1.5.6. *The output H of algorithm* SPECTRAL-SPARSIFY *is a $(1 \pm \epsilon)$-spectral sparsifier with probability at least $1 - 1/n^{c+1}$ for any input graph G that is independent of the random choices of the algorithm.*

Lemma 1.5.7. *With probability at least $1 - 2n^{-c}$, the number of iterations before algorithm* SPECTRAL-SPARSIFY *terminates is*

$$\min\{\lceil \log \rho \rceil, \lceil \log m/((c+1)\log n) \rceil\}.$$

Moreover the size of H is

$$O\left(\sum_{1 \le j \le i} |B_i| + m/\rho + c \log n\right),$$

and the size of the third output of the graph is at most $\max\{O(c \log n), O(m/\rho)\}$.

We conclude that with probability at least $1 - n^{-c}$ our construction yields a $(1 \pm \epsilon)$-spectral sparsifier that also has the properties of Lemma 1.5.7.

Typically, the t-bundle spanners will consist of a polylogarithmic number of spanners of size $O(n \operatorname{poly} \log n)$ and thus the resulting spectral sparsifier will have size $O(n \operatorname{poly} \log n, \epsilon^{-1} + m/\rho)$. In each of the at most $\log n$ iterations the weight of the sampled edges is increased by a factor of 4. Thus, the ratio between the largest and the smallest edge weight in H is at most by a factor of $O(n)$ more than in G, i.e., $O(nW)$.

1.5.3 Decremental Spanner with Monotonicity Property

We first develop the decremental spanner algorithm, which will give us a $(\log n)$-spanner of size $O(n \operatorname{poly}(\log n))$ with a total update time of $O(m \operatorname{poly}(\log n))$. Our algorithm is a careful modification of the dynamic spanner algorithm of Baswana et al. [29] having the following additional monotonicity property: Every time an edge is added to H, it stays in H until it is deleted from G by the adversary. Formally, we will prove the following theorem.

Lemma 1.5.8. *For every $k \geq 2$ and every $0 < \epsilon \leq 1$, there is a decremental algorithm for maintaining a $(1+\epsilon)(2k-1)$-spanner H of expected size $O(k^2 n^{1+1/k} \log n \log_{1+\epsilon} W)$ for an undirected graph G with non-negative edge weights that has an expected total update time of $O(k^2 m \log n)$, where W is the ratio between the largest and the smallest edge weight in G. Additional H has the following property: Every time an edge is added to H, it stays in H until it is deleted from G. The bound on the expected size and the expected running time hold against an oblivious adversary.*

It would be possible to enforce the monotonicity property for any dynamic spanner algorithm by simply overriding the algorithms' decision for removing edges from the spanner before they are deleted from G. Without additional arguments however, the algorithm's bound on the size of the spanner might then not hold anymore. In particular, we do not know how obtain a version of the spanner of Baswana et al. that has the monotonicity property without modifying the internals of the algorithm.

Similar to Baswana et al. [29] we actually develop an algorithm for unweighted graphs and then extend it to weighted graphs as follows. Let W be the ratio of the largest to the smallest edge weight in G. Partition the edges into $\log_{1+\epsilon} W$ subgraphs based on their weights and maintain a $(2k-1)$-spanner ignoring the weights. The union of these spanners will be a $(1+\epsilon)(2k-1)$-spanner of G and the size increases by a factor of $\log_{1+\epsilon} W$ compared to the unweighted version. The update time stays the same as each update in the graph is performed only in one of the $\log_{1+\epsilon} W$ subgraphs. Therefore we assume in the following that G is an unweighted graph.

Algorithm and Running Time

We follow the approach of Baswana et al. and first explain how to maintain a clustering of the vertices and then define our spanner using this clustering.

Clustering. Consider an unweighted undirected graph $G = (V, E)$ undergoing edge deletions. Let $S \subseteq V$ be a subset of the vertices used as cluster centers. Furthermore, consider a permutation σ on the set of vertices V and an integer $i \geq 0$.

The goal is to maintain a clustering $C_{S,\sigma,i}$ consisting of disjoint clusters with one cluster $C_{S,\sigma,i}[s] \subseteq V$ for every $s \in S$. Every vertex within distance i to the vertices in S is assigned to the cluster of its closest vertex in S, where ties are broken according to the permutation σ. More formally, $v \in C_{S,\sigma,i}[s]$ if and only if

- $d_G(v, s) \leq i$ and
- for every $s' \in S \setminus \{s\}$ either
  - $d_G(v, s) < d_G(v, s')$ or
  - $d_G(v, s) = d_G(v, s')$ and $\sigma(s) < \sigma'(s)$.

Observe that each cluster $C_{S,\sigma,i}[s]$ of a vertex $s \in S$ can be organized as a tree consisting of shortest paths to s. We demand that in this tree every vertex v chooses the parent that comes first in the permutation σ among all candidates (i.e., among the vertices that are in the same cluster $C_i[s]$ as v and that are at distance $d(v, s) - 1$ from s).[3] These trees of the clusters define a forest $F_{S,\sigma,i}$ that we wish to maintain together with the clustering $C_{S,\sigma,i}$.

Using a modification of the Even-Shiloach algorithm [75] all the cluster trees of the clustering C_i together can be maintained in total time $O(im \log n)$.

Theorem 1.5.9 ([29])**.** *Given a graph $G = (V, E)$, a set $S \subseteq V$, a random permutation σ of V, and an integer $i \geq 0$, there is a decremental algorithm for maintaining the clustering $C_{S,\sigma,i}$ and the corresponding forest $F_{S,\sigma,i}$ of partial shortest path trees from the cluster centers in expected total time $O(mi \log n)$.*

Note that we deviate from the original algorithm of Baswana et al. by choosing the parent in the tree of each cluster according to the random permutation. In the algorithm of

[3] Using the permutation to choose a random parent is not part of the original construction of Baswana et al.

Baswana et al. the parents in these trees were chosen arbitrarily. However, it can easily be checked that running time guarantee of Theorem 1.5.9 also holds for our modification.

The running time analysis of Baswana et al. hinges on the fact that the expected number of times a vertex changes its cluster is $O(i \log n)$.

Lemma 1.5.10 ([29])**.** *For every vertex v the expected number of times v changes its cluster in $C_{S,\sigma,i}$ is at most $O(i \log n)$.*

By charging time $O(\boldsymbol{d}(v))$ to every change of the cluster of v and every increase of the distance from v to S (which happens at most i times), Baswana et al. get a total update time of $O(im \log n)$ over all deletions in G. For our version of the spanner that has the monotonicity property we additionally need the following observation whose proof is similar to the one of the lemma above.

Lemma 1.5.11. *For every vertex v the expected number of times v changes its parent in $F_{S,\sigma,i}$ is at most $O(i \log n)$.*

Proof. Remember that we assume the adversary to be oblivious, which means that the sequence of deletions is independent of the random choices of our algorithm. We divide the sequence of deletions into phases. For every $1 \leq l \leq i$ the l-th phase consists of the (possibly empty) subsequence of deletions during which the distance from v to S is exactly l, i.e., $d_G(v, S) = l$.

Consider first the case $l \geq 2$. We will argue about possible 'configurations' (s, u) such that v is in the cluster of s and u is the parent of v that might occur in phase l. Let $(s_1, u_1), (s_2, u_2), \ldots, (s_{t^{(l)}}, u_{t^{(l)}})$ (where $t^{(l)} \leq n^2$) be the sequence of all pairs of vertices such that, at the beginning of phase l, for every $1 \leq j \leq t^{(l)}$, s_j is at distance l from v and u_j is a neighbor of v. The pairs (s_i, u_i) in this sequence are ordered according to the point in phase l at which they cease to be possible configurations, i.e., at which either the distance of s_i to v increases to more than l or u is not a neighbor of v anymore.

Let $A_j^{(l)}$ denote the event that, at some point during phase l, v is in the cluster of s_j and u_j is the parent of v. The expected number of times v changes its parent in $F_{S,\sigma,i}$ during phase l is equal to the expected number of j's such that event $A_j^{(l)}$ takes place. Let $B_j^{(l)}$ denote the event that (s_j, u_j) is lexicographically first among all pairs $(s_j, u_j), \ldots, (s_t, u_{t^{(l)}})$ under the permutation σ, i.e., for all $j \leq j' \leq t^{(l)}$ either $\sigma(s_j) \leq \sigma(s_{j'})$ or $\sigma(s_j) = \sigma(s_{j'})$ and $\sigma(u_j), \sigma(u_{j'})$. Observe that $\mathbb{P}\left[A_j^{(l)}\right] \leq \mathbb{P}\left[B_j^{(l)}\right]$ because the event $A_j^{(l)}$ can only take place if the event $B_j^{(l)}$ takes place. Furthermore, $\mathbb{P}\left[B_j^{(l)}\right] = 1/(t^{(l)} - j + 1)$ as every pair of (distinct) vertices has the same probability of being first in the lexicographic order induced by σ. Thus, by linearity of expectation, the number of times v changes its parent in $F_{S,\sigma,i}$ during phase l is at most

$$\sum_{j=1}^{t^{(l)}} \mathbb{P}\left[A_j^{(l)}\right] \leq \sum_{j=1}^{t^{(l)}} \mathbb{P}\left[B_j^{(l)}\right] = \sum_{j=1}^{t^{(l)}} \frac{1}{t^{(l)} - j + 1} = \sum_{j=1}^{t^{(l)}} \frac{1}{j} = O(\log t^{(l)}) = O(\log n)\,.$$

In the second case $l = 1$, a slightly simpler argument bounds the number of times v changes its parent (which is equal to the number of times v changes its cluster) by ordering the neighbors of v in the order of deleting their edge to v. This is the original argument of Baswana et al. [29] of Lemma 1.5.10. We therefore also get that the number of times v changes its parent in $F_{S,\sigma,i}$ in phase 1 is at most $O(\log n)$.

We now sum up the expected number of changes during all phases, and, by linearity of expectation, get that the number of times v changes its parent in $F_{S,\sigma,i}$ is at most $O(i \log n)$.

□

Spanner. Let $2 \leq k \leq \log n$ be a parameter of the algorithm. At the initialization, we first create a sequence of sets $V = S_0 \supseteq S_1 \supseteq \ldots \supseteq S_k = \emptyset$ by obtaining S_{i+1} from sampling each vertex of S_i with probability $n^{-1/k}$. Furthermore, we pick a random permutation σ of the vertices in V.

We use the algorithm of Theorem 1.5.9 to maintain, for every $1 \leq i \leq k$, the clustering $C_i \stackrel{\text{def}}{=} C_{S_i,\sigma}$ together with the forest $F_i \stackrel{\text{def}}{=} F_{S_i,\sigma}$. Define the set V_i as $V_i = \{v \in V \mid$

$d_G(v, S_i) \leq i\}$, i.e., the set of vertices that are at distance at most i to some vertex of S_i. Observe that the vertices in V_i are exactly those vertices that are contained in some cluster $C_i[s]$ of the clustering C_i. For every vertex $v \in V_i$ (where $C_i[s]$ is the cluster of v) we say that a cluster $C_i[s']$ (for some $s' \in S_i \setminus \{s\}$) is *neighboring* to v if G contains an edge (v, v') such that $v' \in C_i[s']$.

Our spanner H consists of the following two types of edges:

1. For every $1 \leq i \leq k$, H contains all edges of the forest F_i consisting of partial shortest path trees from the cluster centers.

2. For every $1 \leq i \leq k$, every vertex $v \in V_i \setminus V_{i+1}$ (contained in some cluster $C_i[s]$), and every neighboring cluster $C_i[s']$ of v, H contains *one* edge to $C_i[s']$, i.e., one edge (v, v') such that $v' \in C_i[s']$.

The first type of edges can be maintained together with the spanning forests of the clustering algorithm of Theorem 1.5.9. The second type of edges can be maintained with the following update rule: Every time the clustering of a vertex $v \in V_i \setminus V_{i+1}$ changes, we add to H *one* edge to each neighboring cluster. Every time such a 'selected' edge is deleted from G, we replace it with another edge to this neighboring cluster until all of them are used up.

We now enforce the monotonicity property mentioned above in the straightforward way. Whenever we have added an edge to H, we only remove it again from H when it is also deleted from G. We argue below that this makes the size of the spanner only slightly worse than in the original construction of Baswana et al.

Stretch and Size

We now prove the guarantees on the stretch and size of H. The stretch argument is very similar to the ones of Baswana et al. We include it here for completeness. In the stretch argument we need stronger guarantees than Baswana et al. as we never remove edges from H, unless they are deleted from G as well.

Lemma 1.5.12 ([29]). *H is a $(2k-1)$-spanner of G.*

Proof. Consider any edge (u, v) of the current graph G and the first j such that u and v are both contained in V_j and at least one of u or v is not contained in V_{j+1}. Without loss of generality assume that $u \notin V_{j+1}$. Since $v \in V_j$, we know that v is contained in some cluster $C_j[s]$ and because of the edge (u, v) this cluster is neighboring to u. Similarly, the cluster of u is neighboring to v. Consider the vertex out of u and v that has changed its cluster within C_i most recently (or take any of the two if both of them haven't changed their cluster since the initialization). Assume without loss of generality that this vertex was u. Then $C_i[s]$ has been a neighboring cluster of u at the time the cluster of u changed, and thus, the spanner H contains some edge (u, v') such that $v' \in C_j[s]$. Using the cluster tree of $C_j[s]$ we find a path from v' to v via s of length at most $2i$ in H. Thus, H contains a path from u to v of length at most $2i + 1 \leq 2k - 1$ as desired. □

Lemma 1.5.13. *The number of edges of H is $O(k^2 n^{1+1/k} \log n)$ in expectation.*

Proof. Consider the first type of edges which are the ones stemming from the partial shortest path trees from the cluster centers. We charge to each vertex v a total of $O(k^2 \log n)$ edges given by all of v's parents in the partial shortest path trees from the cluster centers over the course of the algorithm. For every $1 \leq i \leq k$, we know by Lemma 1.5.11 that the parent of v in F_i changes at most $O(i \log n)$ times in expectation, which gives an overall bound of $O(k^2 \log n)$.

We get the bound on the second type of edges by charging to each vertex v a total of $O(k^2 n^{1/k} \log n)$ edges. Consider a vertex $v \in V_i \setminus V_{i+1}$ for some $0 \leq i \leq k-1$. The number of neighboring clusters of v is equal to the number of vertices of S_i that are at distance exactly $i+1$ from v. Since $v \notin V_{i+1}$ the number of such vertices is $n^{1/k}$ in expectation. Thus, whenever a vertex $v \in V_i \setminus V_{i+1}$ changes its cluster in C_i we can charge $n^{1/k}$ to v_i to pay for the $n^{1/k}$ edges to neighboring clusters. As v changes its cluster in C_i $O(i \log n)$ times by Lemma 1.5.10 and there are k clusterings, the total number of edges of the second type

contained in H is $O(k^2 n^{1+1/k} \log n)$. Note that are allowed to multiply the two expectations because the random variables in question are independent.

The overall bound of $O(k^2 n^{1+1/k} \log n)$ on the expected number of edges follows from the linearity of expectation. □

1.5.4 Decremental Spectral Sparsifier

In the following we explain how to obtain a decremental algorithm for maintaining a spectral sparsifier using the template of Section1.5.2. Internally we use our decremental spanner algorithm of Section 1.5.3. It is conceptually important for our approach to first develop a decremental algorithm, that is turned into a fully dynamic algorithm in Section 1.5.5. We follow the template of Section 1.5.2 by first showing how to maintain t-bundle spanners under edge deletions, and then giving decremental implementations of LIGHT-CUT-SPARSIFY and CUT-SPARSIFY.

The overall algorithm will use multiple instances of the dynamic spanner algorithm, where outputs of one instance will be used as the input of the next instance. We will do so in a strictly hierarchical manner which means that we can order the instances in a way such that the output of instance i only affects instances $i + 1$ and above. In this way it is guaranteed that the updates made to instance i are independent of the internal random choices of instance i, which means that each instance i is running in the oblivious-adversary setting required for Section 1.5.3.

Decremental t-Bundle Spanners

We first show how to maintain a t-bundle $\log n$-spanner under edge deletions for some parameter t. Using the decremental spanner algorithm of Section 1.5.8 with $k = \lfloor (\log n)/4 \rfloor$ and $\epsilon = 1$ we maintain a sequence $H_1, \ldots H_t$ of $\log n$-spanners by maintaining H_i as the spanner of $G \setminus \bigcup_{1 \leq j \leq i-1} H_j$. Here we have to argue that this is legal in the sense that every instance of the algorithm of Lemma 1.5.8 is run on a graph that only undergoes edge

deletions.

Lemma 1.5.14. *If no edges are ever inserted into G after the initialization, then this also holds for $G \setminus \bigcup_{1 \leq j \leq i-1} H_j$ for every $1 \leq i \leq t+1$.*

Proof. The proof is by induction on i. The claim is trivially true for $i = 1$ by the assumption that there are only deletions in G. For $i \geq 2$ we the argument uses the monotonicity property of the dynamic algorithm for maintaining the spanner H_{i-1}. By the induction hypothesis we already know that no edges are ever added to the graph $G \setminus \bigcup_{1 \leq j \leq i-2} H_j$. Therefore the only possibility of an edge being added to $G \setminus \bigcup_{1 \leq j \leq i-1} H_j$ would be to remove an edge e from H_{i-1}. However, by the monotonicity property, when e is removed from H_{i-1}, it is also deleted from G. Thus, e will not be inserted into $G \setminus \bigcup_{1 \leq j \leq i-2} H_j$. □

Our resulting t-bundle $\log n$-spanner then is $B = \bigcup_{1 \leq i \leq t} H_i$, the union of all these spanners. Since the $H_i's$ are disjoint the edges of B can be maintained in the obvious way by observing all changes to the $H_i's$. By our choice of parameters, $n^{1/k} = O(1)$ and thus the expected size of B is $O(tn \log^2 n \log W)$. Observe that Lemma 1.5.14 implies that no edges will ever be inserted into the complement $G \setminus B$, which will be relevant for our application in the spectral sparsifier algorithm. We can summarize the guarantees of our decremental t-bundle spanner algorithm as follows.

Lemma 1.5.15. *For every $t \geq 1$, there is a decremental algorithm for maintaining a t-bundle $\log n$-spanner B of expected size $O(tn \log^2 n \log W)$ for an undirected graph G with non-negative edge weights that has an expected total update time of $O(tm \log^3 n)$, where W is the ratio between the largest and the smallest edge weight in G. Additional B has the following property: After the initialization, no edges are ever inserted into the graph $G \setminus B$. The bound on the expected size and the expected running time hold against an oblivious adversary.*

Dynamic Implementation of LIGHT-SPECTRAL-SPARSIFY

We now show how to implement the algorithm LIGHT-SPECTRAL-SPARSIFY decrementally for a graph G undergoing edge deletions.

For this algorithm we set $t = \lceil 12(c+3)\alpha\epsilon^{-2}\ln n\rceil$. Note that this value is slightly larger than the one proposed in the static pseudocode of Figure 1.1. For the sparsification proof in Section 1.5.2 we have to argue that by our choice of t certain events happen with high probability. In the dynamic algorithm we need ensure the correctness for up to n^2 versions of the graph, one version for each deletion in the graph. By increasing the multiplicative constant in t by 2 (as compared to the static proof of Section 1.5.2) all desired events happen with high probability for all, up to n^2, versions of the graph by a union bound.

The first ingredient of the algorithm is to maintain a t-bundle $\log n$-spanner B of G under edge deletions using the algorithm of Lemma 1.5.15. We now explain how to maintain a graph H' – with the intention that H' contains the sampled non-bundle edges of $G \setminus B$ – as follows: At the initialization, we determine the graph H' by sampling each edge of $G \setminus B$ with probability $1/4$ and adding it to H' with weight $4w_G(e)$. We then maintain H' under the edge deletions in G using the following update rules:

After every deletion in G we first propagate the update to the algorithm for maintaining the t-bundle spanner B, possibly changing B to react to the deletion. We then check whether the deletion in G and the change in B cause an deletion in the complement graph $G \setminus B$. Whenever an edge e is deleted from $G \setminus B$, it is removed from H'. Note that by Lemma 1.5.15 no edge is ever inserted into $G \setminus B$. We now simply maintain the graph H as the union of B and H' and make it the first output of our algorithm; the second output is B.

By the update rules above (and the increased value of t to accommodate for the increased number of events), this decremental algorithm imitates the static algorithm of Figure 1.1 and for the resulting graph H we get the same guarantees as in Lemma 1.5.5. The total update time of our decremental version of LIGHT-SPECTRAL-SPARSIFY is $O(tm\log^3 n)$, as it is dominated by the time for maintaining the t-bundle $\log n$-spanner B.

As an additional property we get that no edge is ever added to the graph $H' = H \setminus B$. Furthermore, for all edges added to H' weights are always increased by the same factor. Therefore the ratio between the largest and the smallest edge weight in H' will always be bounded by W, which is the value of this quantity in G (before the first deletion).

Dynamic Implementation of SPECTRAL-SPARSIFY

Finally, we show how to implement the algorithm SPECTRAL-SPARSIFY decrementally for a graph G undergoing edge deletions.

We set $k = \lceil \log \rho \rceil$ as in the pseudocode of Figure 1.2 and maintain k instances of the dynamic version of LIGHT-SPECTRAL-SPARSIFY above. We maintain the k graphs $G_0, \ldots, G_k$, $B_1, \ldots, B_k$, and $H_1, \ldots, H_k$ as in the pseudocode. For every $1 \leq i \leq k$ we maintain H_i and B_i as the two results of running the decremental version of LIGHT-SPECTRAL-SPARSIFY on G_{i-1} and maintain G_i as the graph $H_i \setminus B_i$. As argued above (for H' in Section 1.5.4), no edge is ever added to $G_i = H_i \setminus B_i$ for every $1 \leq i \leq k$ and we can thus use our purely decremental implementation of LIGHT-SPECTRAL-SPARSIFY.

At the initialization, we additionally count the number of edges of every graph G_i and ignore every graph G_i with less than $(c+1)\ln n$ edges. Formally we set k maximal such that G_k has at least $(c+1)\ln n$ edges.

The output of our algorithm is the graph $H = \bigcup_{i=1}^{k} B_i \cup G_k$. Now by the same arguments as for the static case, H gives the same guarantees as in Lemma 1.5.6 and 1.5.7. Thus, by our choices of k and t, H is a $(1 \pm \epsilon)$-spectral sparsifier of size $O(c\epsilon^{-2} \log^3 \rho \log^4 n \log W + m\rho^{-1})$. As the total running time is dominated by the running time of the k instances of the decremental algorithm for LIGHT-SPECTRAL-SPARSIFY, the total update time is $O(cm\epsilon^{-2} \log^3 \rho \log^5 n)$. The guarantees of our decremental sparsifier algorithm can be summarized as follows.

Lemma 1.5.16. *For every $0 < \epsilon \leq 1$, every $1 \leq \rho \leq m$, and every $c \geq 1$, there is a decremental algorithm for maintaining, with probability at least $1 - 1/n^c$ against an*

oblivious adversary, a $(1 \pm \epsilon)$*-spectral sparsifier* H *of size* $O(c\epsilon^{-2} \log^3 \rho \log^4 n \log W + m\rho^{-1})$ *for an undirected graph* G *with non-negative edge weights that has a total update time of* $O(cm\epsilon^{-2} \log^3 \rho \log^5 n)$*, where* W *is the ratio between the largest and the smallest edge weight in* G*.*

1.5.5 Turning Decremental Spectral Sparsifier into Fully Dynamic Spectral Sparsifier

We use a well-known reduction to turn our decremental algorithm into a fully dynamic algorithm.

Lemma 1.5.17. *Given a decremental algorithm for maintaining a* $(1 \pm \epsilon)$*-spectral (cut) sparsifier of size* $S(m, n, W)$ *for an undirected graph with total update time* $m \cdot T(m, n, W)$*, there is a fully dynamic algorithm for maintaining a* $(1 \pm \epsilon)$*-spectral (cut) sparsifier of size* $O(S(m, n, W) \log n)$ *with amortized update time* $O(T(m, n, W) \log n)$*.*

Together with Lemma 1.5.16 this immediately implies Theorem 1.5.1. A similar reduction has been used by Baswana et al. [29] to turn their decremental spanner algorithm into a fully dynamic one. The only additional aspect we need is the lemma below on the decomposability of spectral sparsifiers. We prove this property first and then give the reduction, which carries over almost literally from [29].

Lemma 1.5.18 (Decomposability)**.** *Let* $G = (V, E)$ *be an undirected weighted graph, let* $E_1, \ldots, E_k$ *be a partition of the set of edges* E*, and let, for every* $1 \leq i \leq k$*,* H_i *be a* $(1 \pm \epsilon)$*-spectral sparsifier of* $G_i = (V, E_i)$*. Then* $H = \bigcup_{i=1}^{k} H_i$ *is a* $(1 \pm \epsilon)$*-spectral sparsifier of* G*.*

Proof. Because H_i is a spectral sparsifier of G_i, for any vector x and $i = 1, \ldots, k$ we have

$$(1 - \epsilon) x^T \mathcal{L}_{H_i} x \leq x^T \mathcal{L}_{G_i} x \leq (1 + \epsilon) x^T \mathcal{L}_{H_i} x$$

Summing these k inequalities, we get that

$$(1-\epsilon)x^T\mathcal{L}_H x \leq x^T\mathcal{L}_G x \leq (1+\epsilon)x^T\mathcal{L}_H x,$$

which by definition means that H is a $(1\pm\epsilon)$-spectral sparsifier of H. □

Proof of Lemma 1.5.17. Set $k = \lceil \log(n^2) \rceil$. For each $1 \leq i \leq k$, we maintain a set $E_i \subseteq E$ of edges and an instance A_i of the decremental algorithm running on the graph $G_i = (V, E_i)$. We also keep a binary counter C that counts the number of insertions modulo n^2 with the least significant bit in C being the right-most one.

A deletion of some edge e is carried out by simply deleting e from the set E_i it is contained in and propagating the deletion to instance A_i of the decremental algorithm.

An insertion of some edge e is carried out as follows. Let j be the highest (i.e., left-most) bit that gets flipped in the counter when increasing the number of insertions. Thus, in the updated counter the j-th bit is 1 and all lower bits (i.e., bits to the right of j) are 0. We first add the edge e as well as all edges in $\bigcup_{i=1}^{j-1} E_i$ to E_j. Then we set $E_i = \emptyset$ for all $1 \leq i \leq j-1$. Finally, we re-initialize the instance A_j on the new graph $G_j = (V, E_j)$.

We know bound the total update time for each instance A_i of the decremental algorithm. First, observe that the i-th bit of the binary counter is reset after every 2^i edge insertions. A simple induction then shows that at any time $E_i \leq 2^i$ for all $1 \leq i \leq k$. Now consider an arbitrary sequence of updates of length ℓ. The instance A_i is re-initialized after every 2^i insertions. It will therefore be re-initialized at most $\ell/2^i$ times. For every re-initialization we pay a total update time of $|E_i| \cdot T(|E_i|, n, W) \leq 2^i T(m, n, W)$. For the entire sequence of ℓ updates, the total time spent for instance A_i is therefore $(\ell/2^i) \cdot 2^i T(m, n, W) = \ell \cdot T(m, n, W)$. Thus we spend total time $O(\ell \cdot T(m, n, W) \log n)$ for the whole algorithm, which amounts to an amortized update time of $O(T(m, n, W) \log n)$. □

1.6 Dynamic Cut Sparsifier

In this section we give an algorithm for maintaining a cut sparsifier under edge deletions and insertions with polylogarithmic worst-case update time. The main result of this section is as follows.

Theorem 1.6.1. *There exists a fully dynamic randomized algorithm with polylogarithmic update time for maintaining a $(1 \pm \epsilon)$-cut sparsifier H of a graph G, with probability at least $1 - n^{-c}$ for any $0 < \epsilon \leq 1$ and $c \geq 1$. Specifically, the algorithm either has worst-case update time*

$$O(c\epsilon^{-2} \log^2 \rho \log^5 n \log W)$$

or amortized update time

$$O(c\epsilon^{-2} \log^2 \rho \log^3 n \log W)$$

and the size of H is

$$O(cn\epsilon^{-2} \log^2 \rho \log n \log W + m\rho^{-1}),$$

where $1 \leq \rho \leq m$ is a parameter of choice. Here, W is the ratio between the largest and the smallest edge weight in G. The ratio between the largest and the smallest edge weight in H is at most $O(nW)$.

By running the algorithm with basically $\rho = m$ we additionally get that H has low arboricity, i.e., it can be partitioned into a polylogarithmic number of trees. We will algorithmically exploit the low arboricity property in Section 1.7 and 1.9.

Corollary 1.6.2. *There exists a fully dynamic randomized algorithm with polylogarithmic update time for maintaining a $(1 \pm \epsilon)$-cut sparsifier H of a graph G, with probability at least $1 - n^{-c}$ for any $0 < \epsilon \leq 1$ and $c \geq 1$. Specifically, the algorithm either has worst-case update time $O(c\epsilon^{-2} \log^7 n \log W)$ or amortized update time $O(c\epsilon^{-2} \log^5 n \log W)$. The arboricity of H is $k = O(c\epsilon^{-2} \log^3 n \log W)$. Here, W is the ratio between the largest and*

the smallest edge weight in G. The ratio between the largest and the smallest edge weight in H is at most $O(nW)$. We can maintain a partition of H into disjoint forests $T_1, \ldots, T_k$ such that every vertex keeps a list of its neighbors together with its degree in each forest T_i. After every update in G at most one edge is added to and at most one edge is removed from each forest T_i.

After giving an overview of our algorithm, we first explain our cut sparsification scheme in a static setting and prove its properties. Subsequently, we show how we can dynamically maintain the edges of such a sparsifier with both amortized and worst-case update times by making this scheme dynamic.

1.6.1 Algorithm Overview

Our Framework. The algorithm is based on the observation that the spectral sparsification scheme outlined above in Section 1.4.1. becomes a cut sparsification algorithm if we simply replace spanners by maximum weight spanning trees (MSTs). This is inspired by sampling according to edge connectivities; the role of the MSTs is to certify lower bounds on the edge connectivities. We observe that the framework does not require us to use exact MSTs. For our t-bundles we can use a relaxed, approximate concept that we call α-MST that. Roughly speaking, an α-MST guarantees a 'stretch' of α in the infinity norm and, as long as it is sparse, does not necessarily have to be a tree.

Similarly to before, we define a t-bundle α-MST B as the union of a sequence of α-MSTs $T_1, \ldots T_t$ where the edges of each tree are removed from the graph before computing the next α-MST. The role of α-MST is to certify uniform lower bounds on the connectivity of edges; these bounds are sufficiently large to allow uniform sampling with a fixed probability.

This process of peeling and sampling is repeated sufficiently often and our cut sparsifier then is the union of all the t-bundle α-MSTs and the non-bundle edges remaining after taking out the last bundle. Thus, the cut sparsifier consists of a polylogarithmic number of α-MSTs and a few (polylogarithmic) additional edges. This means that for α-MSTs based

on spanning trees, our cut sparsifiers are not only sparse, but also have polylogarithmic *arboricity*, which is the minimum number of forests into which a graph can be partitioned.

Simple Fully Dynamic Algorithm. Our approach immediately yields a fully dynamic algorithm by using a fully dynamic algorithm for maintaining a spanning forest. Here we basically have two choices. Either we use the randomized algorithm of Kapron, King, and Mountjoy [24] with polylogarithmic worst-case update time. Or we use the deterministic algorithm of Holm, de Lichtenberg, and Thorup [23] with polylogarithmic amortized update time. The latter algorithm is slightly faster, at the cost of providing only amortized update-time guarantees. A t-bundle 2-MST can be maintained fully dynamically by running, for each of the $\log W$ weight classes of the graph, t instances of the dynamic spanning tree algorithm in a 'chain'.

An important observation about the spanning forest algorithm is that with every update in the graph, at most one edge is changed in the spanning forest: If for example an edge is deleted from the spanning forest, it is replaced by another edge, but no other changes are added to the tree. Therefore a single update in G can only cause one update for each graph $G_i = G \setminus \bigcup_{j=1}^{i-1} T_j$ and T_i. This means that each instance of the spanning forest algorithm creates at most one 'artificial' update that the next instance has to deal with. In this way, each dynamic spanning forest instance used for the t-bundle has polylogarithmic update time. As $t = \mathrm{poly}(\log n)$, the update time for maintaining a t-bundle is also polylogarithmic. The remaining steps of the algorithm can be carried out dynamically in the straightforward way and overall give us polylogarithmic worst-case or amortized update time.

A technical detail of our algorithm is that the high-probability correctness achieved by the Chernoff bounds only holds for a polynomial number of updates in the graph. We thus have to restart the algorithm periodically. This is trivial when we are shooting for an amortized update time. For a worst-case guarantee we can neither completely restart the algorithm nor change all edges of the sparsifier in one time step. We therefore keep

two instances of our algorithm that maintain two sparsifiers of two alternately growing and shrinking subgraphs that at any time partition the graph. This allows us to take a blend of these two subgraph sparsifiers as our end result and take turns in periodically restarting the two instances of the algorithm.

1.6.2 Definitions

We will work with a relaxed notion of an MST, which will be useful when maintaining an exact maximum spanning tree is hard (as is the case for worst-case update time guarantees).

Definition 1.6.3. A subgraph T of an undirected graph G is an α-MST ($\alpha \geq 1$) if for every edge $e = (u, v)$ of G there is a path π from u to v such that $w_G(e) \leq \alpha w_G(f)$ for every edge f on π.

Note that in this definition we do not demand that T is a tree; any subgraph with these properties will be fine. A maximum spanning tree in this terminology is a 1-MST.

Definition 1.6.4. A *t-bundle α-MST* ($t, \alpha \geq 1$) of an undirected graph G is the union $B = \bigcup_{i=1}^{k} T_i$ of a sequence of graphs $T_1, \ldots, T_t$ such that, for every $1 \leq i \leq t$, T_i is an α-MST of $G \setminus \bigcup_{j=1}^{i-1} T_j$.

We can imagine such a t-bundle being obtained by iteratively peeling-off α-MSTs from G.

1.6.3 A Simple Cut Sparsification Algorithm

We begin with algorithm LIGHT-CUT-SPARSIFY in Figure 1.3; this is the core iteration used to compute a sparser cut approximation with approximately half the edges. Algorithm CUT-SPARSIFY in Figure 1.3 is the full sparsification routine.

The properties of these algorithm are given in the following lemmas.

Lemma 1.6.5. *The output H of algorithm* LIGHT-CUT-SPARSIFY *is a $(1 \pm \epsilon)$-cut approximation of the input G, with probability $1 - n^{-c}$.*

LIGHT-CUT-SPARSIFY (G, c, ϵ)

1. $t \leftarrow C_\xi c\alpha \log W \log^2 n/\epsilon^2$
2. Let B be a t-bundle α-MST of G
3. $H := B$
4. **For each** edge $e \in G \setminus B$
   (a) With probability $1/4$ add e to H with $4w_H(e) \leftarrow w_G(e)$
5. **Return** (H, B)

Figure 1.3: LIGHT-CUT-SPARSIFY (G, c, ϵ). We give a dynamic implementation of this algorithm in Section 1.6.4. In particular we dynamically maintain the t-bundle α-MST B which results in a dynamically changing graph $G \setminus B$.

CUT-SPARSIFY (G, c, ϵ)

1. $k \leftarrow \lceil \log \rho \rceil$
2. $G_0 \leftarrow G$
3. $B_0 \leftarrow (V, \emptyset)$
4. **for** $i = 1$ **to** k
   (a) $(H_i, B_i) \leftarrow$ LIGHT-CUT-SPARSIFY$(G, c + 1, \epsilon/(2k))$
   (b) $G_{i+1} \leftarrow H_i \setminus B_i$
   (c) **if** G_{i+1} has less than $(c + 2) \ln n$ edges **then break** (* break loop *)
5. $H \leftarrow \bigcup_{1 \le j \le i} B_j \cup G_{i+1}$
6. **return** $(H, \{B_j\}_{j=1}^i, G_{i+1})$

Figure 1.4: CUT-SPARSIFY (G, c, ϵ) We give a dynamic implementation of this algorithm in Section 1.6.4. In particular we dynamically maintain each H_i and B_i as the result of a dynamic implementation of LIGHT-CUT-SPARSIFY which results in dynamically changing graphs G_i.

We will need a slight generalization of a Theorem in [64].

Lemma 1.6.6. *(generalization of Theorem 1.1 [64]) Let H be obtained from a graph G with weights in $(1/2, 1]$ by independently sampling edge edge e with probability $p_e \geq \rho/\lambda_G(e)$, where $\rho = C_\xi c \log^2 n/4\epsilon^2$, and $\lambda_G(e)$ is the local edge connectivity of edge e, C_ξ is an explicitly known constant. Then H is a $(1 \pm \epsilon)$-cut sparsifier, with probability at least $1 - n^{-c}$.*

Proof. (Sketch) The generalization lies in introducing the parameter c to control the probability of failure. This reflects the standard behavior of Chernoff bounds: increasing the number of samples by a factor of c drives down the failure probability by a factor of n^{-c}. Also, the original theorem assumes that all edges are unweighted, but a standard variant of the Chernoff bound can absorb constant ranges, with a corresponding constant factor increase in the number of samples. Finally, the original theorem is stated with $p_e = \rho/\lambda_G(e)$, but all arguments remain identical if this is relaxed to an inequality. □

Proof. Suppose without loss of generality that the maximum weight in G is 1. We decompose G into $\log W$ edge-disjoint graphs, where G_i consists of the edges with weights in $(2^{-(i+1)}, 2^{-i}]$ plus $B_i = B/2^{-(i+1)}$, where B is the bundle returned by the algorithm.

By definition of the α-MST t-bundle, the connectivity of each edge of $G_i \setminus B_i$ in G_i is at least $4\rho c$, for $c = d \log W$ where ρ is as defined in Lemma 1.6.6. Assume for a moment that all edges in B_i are also in $(2^{-(i+1)}, 2^{-i}]$. Then we can set $p_e = 1$ for each $e \in B_i$ and $p_e = 1/4$ for all other edges, and apply Lemma 1.6.6. In this way we get that H_i is $(1 \pm \epsilon)$-cut sparsifier with probability at least $1 - n^{d \log W}$.

The assumption about B_i can be removed as follows. We observe that one can find a subgraph B_i' of B_i (by splitting weights when needed, and dropping smaller weights), such that B_i' is a t-bundle α-MST of G_i. This follows by the definition of the t-bundle α-MST . We can thus apply the lemma on $G_i' = (G_i \setminus B_i) \cup B_i'$, and get that the sampled graph H_i' is a $(1 \pm \epsilon)$-cut sparsifier. We then observe that $G_i = G_i' \cup (B_i \setminus B_i')$ and $H_i = G_i' \cup (B_i \setminus B_i')$,

from which it follows that H_i is a $(1 \pm \epsilon)$-cut sparsifier of G_i. □

Note: The number of logarithms in LIGHT-CUT-SPARSIFY is not optimal. One can argue that the lower bounds we compute can be used in place of the *strong connectivities* used in [33] and reduce by one the number of logarithms. It is also possible to replace $\log W$ with $\log n$ by carefully re-working some of the details in [33].

We finally have the following Lemmas. The proofs are identical to those for the corresponding Lemmas in Section 1.5, so we omit them.

Lemma 1.6.7. *The output H of algorithm* CUT-SPARSIFY *is a $(1 \pm \epsilon)$-spectral sparsifier of the input G, with probability at least $1 - 1/n^{c+1}$.*

Lemma 1.6.8. *With probability at least $1 - 2n^{-c}$, the number of iterations before algorithm* CUT-SPARSIFY *terminates is*

$$\min\{\lceil \log \rho \rceil, \lceil \log m/((c+1)\log n) \rceil\}.$$

Moreover the size of H is

$$O\left(\sum_{1 \le j \le i} |B_i| + m/\rho + c \log n\right),$$

and the size of the third output of the graph is at most $\max\{O(c \log n), O(m/\rho)\}$.

1.6.4 Dynamic Cut Sparsifier

We now explain how to implement the cut sparsifier algorithm of Section 1.6.3 dynamically. The main building block of our algorithm is a fully dynamic algorithm for maintaining a spanning forest with polylogarithmic update time. We either use an algorithm with worst-case update time, or a slightly faster algorithm with amortized update time. In both algorithms, an insertion might join two subtrees of the forest and after a deletion the forest

is repaired by trying to find a single replacement edge. This strongly bounds the number of changes in the forest after each update.

Theorem 1.6.9 ([24, 76])**.** *There is a fully dynamic deterministic algorithm for maintaining a spanning forest T of an undirected graph G with worst-case update time $O(\log^4 n)$. Every time an edge e is inserted into G, the only potential change to T is the insertion of e. Every time an edge e is deleted from G, the only potential change to T is the removal of e and possibly the addition of at most one other edge to T. The algorithm is correct with high probability against an oblivious adversary.*

Theorem 1.6.10 ([23])**.** *There is a fully dynamic deterministic algorithm for maintaining a minimum spanning forest T of a weighted undirected graph G with amortized update time $O(\log^2 n)$. Every time an edge e is inserted into G, the only potential change to T is the insertion of e. Every time an edge e is deleted from G, the only potential change to T is the removal of e and possibly the addition of at most one other edge to T.*

We first explain how to use these algorithms in a straightforward way to maintain a 2-MST. Subsequently we show how to dynamically implement the procedures LIGHT-CUT-SPARSIFY and CUT-SPARSIFY. The overall algorithm will use multiple instances of a dynamic spanning forest algorithm, where outputs of one instance will be used as the input of the next instance. We will do so in a strictly hierarchical manner which means that we can order the instances in a way such that the output of instance i only affects instances $i+1$ and above. In this way it is guaranteed that the updates made to instance i are independent of the internal random choices of instance i, which means that each instance i is running in the oblivious-adversary setting required for Theorem1.6.9.

Dynamic Maintenance of 2*-MST*

For every $0 \leq i \leq \lfloor \log W \rfloor$, let E_i be the set of edges of weight between 2^i and 2^{i+1}, i.e., $E_i = \{e \in E \mid 2^i \leq w_G(e) < 2^{i+1}\}$, and run a separate instance of the dynamic spanning

forest algorithm for the edges in E_i. For every $0 \leq i \leq \lfloor \log W \rfloor$, let F_i be the spanning forest of the edges in E_i maintained by the i-th instance. We claim that the union of all these trees is a 2-MST of G.

Lemma 1.6.11. *$T = \bigcup_{i=0}^{\lfloor \log W \rfloor} F_i$ is a 2-MST of G.*

Proof. Consider some edge $e = (u, v)$ of G and let i be the (unique) index such that $2^i \leq w_G(e) < 2^{i+1}$. Since F_i is spanning tree of G, there is a path π from u to v in F_i (and thus also in T). Every edge f of π is in the same weight class as e, i.e., $2^i \leq w_G(f) < 2^{i+1}$. Thus, $w_G(e) < 2^{i+1} \leq 2w_G(f)$ as desired. □

Every time an edge e is inserted or deleted, we determine the weight class i of e and perform the update in the i-th instance of the spanning forest algorithm. This 2-MST of size $O(n \log W)$ can thus be maintained with the same asymptotic update time as the dynamic spanning forest algorithm.

We now show how to maintain a t-bundle 2-MST and consequently a $(1 \pm \epsilon)$-cut sparsifier H according to the construction presented in Section 1.6.3. For the t-bundle 2-MST $B = \bigcup_{1 \leq i \leq k} T_i$ we maintain, for every $1 \leq i \leq t$, a 2-MST of $G \setminus \bigcup_{j=1}^{i-1} T_j$. We now analyze how changes to $G \setminus \bigcup_{j=1}^{i-1} T_j$ affect $G \setminus \bigcup_{j=1}^{i} T_j$ (for every $1 \leq i \leq k$):

- Whenever an edge e is inserted into $G \setminus \bigcup_{j=1}^{i-1} T_j$, the 2-MST algorithm either adds e to T_i or not.
  - If e is added to T_i, then $G \setminus \bigcup_{j=1}^{i} T_j$ does not change.
  - If e is not added to T_i, then e is added to $G \setminus \bigcup_{j=1}^{i} T_j$.
- Whenever an edge e is deleted from $G \setminus \bigcup_{j=1}^{i-1} T_j$, either e is contained in T_i or not.
  - If e is contained in T_i, then e is removed from T_i and some other edge f is added to T_i. This edge f is removed from $G \setminus \bigcup_{j=1}^{i} T_j$.[4]

[4]The edge e will not be added to $G \setminus \bigcup_{j=1}^{i} T_j$ because it is removed from both $G \setminus \bigcup_{j=1}^{i-1} T_j$ and T_i.

– If e is not contained in T_i, then e is removed from $G \setminus \bigcup_{j=1}^{i} T_j$.

Thus, every change to $G \setminus \bigcup_{j=1}^{i-1} T_j$ results in at most one change to $G \setminus \bigcup_{j=1}^{i} T_j$. Consequently, a single update to G results to at most one update in each instance of the dynamic MST algorithm. For every update in G we therefore incur an amortized update time of $O(t \log^4 n)$. Thus, we can summarize the guarantees for maintaining a t-bundle 2-MST as follows.

Corollary 1.6.12. *There are fully dynamic algorithms for maintaining a t-bundle 2-MST B (where $t \geq 1$ is an integer) of size $O(tn \log W)$ with worst-case update time $O(t \log^4 n)$ or amortized update time $O(t \log^2 n)$, respectively. After every update in G, the graph $G \setminus B$ changes by at most one edge.*

Dynamic Implementation of LIGHT-CUT-SPARSIFY

For this algorithm we set $t = (C_\xi + 2) d\alpha\epsilon^{-2} \log W \log^2 n$. Note that this value is slightly larger than the one proposed in Figure 1.3. For the sparsification proof in Section 1.6.3 we have to argue that by our choice of t certain events happen with high probability. In the dynamic algorithm we need ensure the correctness for a polynomial number of versions of the graph, one version for each update made to the graph. We show in Lemma 1.6.5 that it is sufficient to be correct for up to $4n^2$ updates to the graph, as then we can extend the algorithm to an arbitrarily long sequence of updates. By making t slightly large than in the static proof of Section 1.6.3 all the desired events happen with high probability for all $4n^2$ versions of the graph by a union bound.

The first ingredient of the algorithm is to dynamically maintain a t-bundle 2-MST B using the algorithm of Lemma 1.6.12 above. We now explain how to maintain a graph H' – with the intention that H' contains the sampled non-bundle edges of $G \setminus B$ – as follows: After every update in G we first propagate the update to the algorithm for maintaining the t-bundle 2-MST B, possibly changing B to react to the update. We then check whether the update in G and the change in B cause an update in the complement graph $G \setminus B$.

- Whenever an edge is inserted into $G \setminus B$, it is added to H' with probability $1/4$ and weight $4w_G(e)$.
- Whenever an edge e is deleted from $G \setminus B$, it is removed from H'.

We now simply maintain the graph H as the union of B and H' and make it the first output of our algorithm; the second output is B.

By the update rules above (and the increased value of t to accommodate for the increased number of events), this dynamic algorithm imitates the static algorithm of Figure 1.3 and for the resulting graph H we get the same guarantees as in Lemma 1.6.5. The update time of our dynamic version of LIGHT-SPECTRAL-SPARSIFY is $O(t \log^4 n)$ worst-case and $O(t \log^2 n)$ worst-case, as it is dominated by the time for maintaining the t-bundle 2-MST B.

As an additional property we get that with every update in G at most one change is performed to $H' = H \setminus B$. Furthermore, for all edges added to H' weights are always increased by the same factor. Therefore the ratio between the largest and the smallest edge weight in H' will always be bounded by W, which is the value of this quantity in G (before the first deletion).

Dynamic Implementation of CUT-SPARSIFY

We set $k = \lceil \log \min(\rho, m/((c+2)\log n)) \rceil$ and maintain k instances of the dynamic version of LIGHT-CUT-SPARSIFY above, using the other parameters just like in the pseudo-code of Figure 1.4. By this choice of k we ensure that we do not have to check the breaking condition in the pseudo-code explicitly, which is more suited for a dynamic setting where the number of edges in the maintained subgraphs might grow and shrink.

We maintain the k graphs $G_0, \ldots, G_k$, $B_1, \ldots, B_k$, and $H_1, \ldots, H_k$ as in the pseudocode. For every $1 \leq i \leq k$ we maintain H_i and B_i as the two results of running the dynamic version of LIGHT-CUT-SPARSIFY on G_{i-1} and maintain G_i as the graph $H_i \setminus B_i$.

The output of our algorithm is the graph $H = \bigcup_{i=1}^{k} B_i \cup G_k$. Note that, by our choice of k, G_k has at most $\max(m/\rho, (c+2)\log n)$ edges. Now by the same arguments as for the

static case, H gives the same guarantees as in Lemma 1.6.7 and 1.6.8 for up to a polynomial number of updates (here at most $4n^2$) in the graph.

As argued above (for H' in Section 1.6.4), every update in G_{i-1} results in at most one change to $G_i = H_i \setminus B_i$ for every $1 \leq i \leq k$. By an inductive argument this means that every update in G results in at most one change to G_i for every $1 \leq i \leq k$. As each instance of the dynamic LIGHT-CUT-SPARSIFY algorithm has update time $O(t \log^4 n)$ worst-case or $O(t \log^2 n)$ amortized, this implies that our overall algorithm has update time $O(kt \log^4 n)$ or $O(kt \log^2 n)$, respectively. Together with Lemma 1.6.14 in Section 1.6.3, we have proved Theorem 1.6.1 stated at the beginning of this section.

In Lemma 1.6.2 we additionally claim that for $\rho = m$ we obtain a sparsifier with polylogarithmic arboricity. This is true because the cut sparsifier H mainly consists of a collection of bundles, which in turn consists of a collection of trees. In total, H consists of $O(tk \log W)$ trees and $O(c \log n)$ remaining edges in G_k, each of which can be seen as a separate tree. Furthermore we can maintain the collection of trees explicitly with appropriate data structures for storing them.

1.6.5 Handling Arbitrarily Long Sequences of Updates

The high-probability guarantees of the algorithm above only holds for a polynomially bounded number of updates. We now show how to extend it to an arbitrarily long sequence of updates providing the same asymptotic update time and size of the sparsifier. We do this by concurrently running two instances of the dynamic algorithm that periodically take turns in being restarted, which is a fairly standard approach for such situations. The only new aspect necessary for our purposes is that both instances explicitly maintain a sparsifier and when taking turns we cannot simply replace all the edges of one sparsifier with the edges of the other sparsifier as processing all these edges would violate the worst-case update time guarantee. For this reason we exploit the decomposability of graph sparsifiers and maintain a 'blend' of the two sparsifiers computed by the concurrent instances of the dynamic algorithm.

This step is not necessary for other dynamic problems such as connectivity where we only have to make sure that the query is delegated to the currently active instance.

Lemma 1.6.13 (Decomposability)**.** *Let $G = (V, E)$ be an undirected weighted graph, let $E_1, \ldots, E_k$ be a partition of the set of edges E, and let, for every $1 \leq i \leq k$, H_i be a $(1 \pm \epsilon)$-cut sparsifier of $G_i = (V, E_i)$. Then $H = \bigcup_{i=1}^{k} H_i$ is a $(1 \pm \epsilon)$-cut sparsifier of G.*

Proof. Let U be a cut in G. First observe that

$$w_G(\partial_G(U)) = w_G(\bigcup_{i=1}^{k} \partial_{G_i}(U)) = \sum_{i=1}^{k} w_G(\partial_{G_i}(U)) = \sum_{i=1}^{k} w_{G_i}(\partial_{G_i}(U))$$

and similarly $w_H(\partial_H(U)) = \sum_{i=1}^{k} w_{H_i}(\partial_{H_i}(U))$. Now since

$$(1-\epsilon) w_{H_i}(\partial_{H_i}(U)) \leq w_{G_i}(\partial_{G_i}(U)) \leq (1+\epsilon) w_{H_i}(\partial_{H_i}(U))$$

for every $1 \leq i \leq k$, we have

$$\begin{aligned}(1-\epsilon) w_H(\partial_H(U)) = (1-\epsilon) \sum_{i=1}^{k} w_{H_i}(\partial_{H_i}(U)) \leq \sum_{i=1}^{k} w_{G_i}(\partial_{G_i}(U)) = w_G(\partial_G(U)) \\ = \cdots \leq (1+\epsilon) w_H(\partial_H(U))\,.\end{aligned}$$

□

Lemma 1.6.14. *Assume there is a fully dynamic algorithm for maintaining a $(1 \pm \epsilon)$-cut (spectral) sparsifier of size at most $S(m, n, W)$ with worst-case update time $T(m, n, W)$ for up to $4n^2$ updates in G. Then there also is a fully dynamic algorithm for maintaining a $(1 \pm \epsilon)$-cut (spectral) sparsifier of size at most $O(S(m, n, W))$ with worst-case update time $O(T(m, n, W))$ for an arbitrary number of updates.*

Proof. We exploit the decomposability of cut sparsifiers. We maintain a partition of G into two disjoint subgraphs G_1 and G_2 and run two instances A_1 and A_2 of the dynamic

algorithm on G_1 and G_2, respectively. These two algorithms maintain a $(1 \pm \epsilon)$-sparsifier of H_1 of G_1 and a $(1 \pm \epsilon)$-sparsifier H_2 of G_2. By the decomposability stated in Lemma 1.6.13 and 1.5.18, the union $H \stackrel{\text{def}}{=} H_1 \cup H_2$ is a $(1 \pm \epsilon)$-sparsifier of $G = G_1 \cup G_2$.

We divide the sequence of updates into phases of length n^2 each. In each phase of updates one of the two instances A_1, A_2 is in the state *growing* and the other one is in the state *shrinking*. A_1 and A_2 switch their states at the end of each phase. In the following we describe the algorithm's actions during one phase. Assume without loss of generality that, in the phase we are fixing, A_1 is growing and A_2 is shrinking.

At the beginning of the phase we restart the growing instance A_1. We will orchestrate the algorithm in such a way that at the beginning of the phase G_1 is the empty graph and $G_2 = G$. After every update in G we execute the following steps:

1. If the update was the insertion of some edge e, then e is added to the graph G_1 and this insertion is propagated to the *growing* instance A_1.

2. If the update was the deletion of some edge e, then e is removed from the graph G_i it is contained in and this deletion is propagated to the corresponding instance A_i.

3. In addition to processing the update in G, if G_2 is non-empty, then one arbitrary edge e is first removed from G_2 and deleted from instance A_2 and then added to G_1 and inserted into instance A_1.

Observe that these rules indeed guarantee that G_1 and G_2 are disjoint and together contain all edges of G. Furthermore, since the graph G_2 of the shrinking instance has at most n^2 edges at the beginning of the phase, the length of n^2 updates per phase guarantees that G_2 is empty at the end of the phase. Thus, the growing instance always starts with an empty graph G_1.

As both H_1 and H_2 have size at most $S(n, m, W)$, the size of $H = H_1 \cup H_2$ is $O(S(n, m, W))$. With every update in G we perform at most 2 updates in each of A_1 and A_2. It follows that the worst-case update time of our overall algorithm is $O(T(m, n, W))$.

Furthermore since each of the instances A_1 and A_2 is restarted every other phase, each instance of the dynamic algorithm sees at most $4n^2$ updates before it is restarted. □

1.7 Application of Dynamic Cut Sparsifier: Undirected Bipartite Min-Cut

We now utilize our sparsifier data structure to maintain a $(2+\epsilon)$-approximate st-min-cut in amortized $O(\text{poly}(\log n, \epsilon^{-1}))$ time per update. In this section, we will define several tools that are crucial for the better analyses in Sections 1.8 and 1.9.

This result is a weaker form of Theorem 1.2.3 with an approximation factor of $2+\epsilon$ instead of $1+\epsilon$. The main result that we will show in this section is:

Theorem 1.7.1. *For every $0 < \epsilon \leq 1$, there is a fully dynamic algorithm for maintaining a $(2+\epsilon)$-approximate minimum cut in an unweighted undirected graph that's a bipartite graph with source/sink s and t attached to each of the partitions with amortized update time $O(\text{poly}(\log n, \epsilon^{-1}))$.*

To add motivation for solving this problem, we would like to point out that there are examples in which the maximum $s-t$ flow is much larger than the minimum vertex cover, and we cannot simply consider the problem as finding a maximum matching in $G_{A,B}$. Specifically, let $A = A_{k^2} \cup A_k$ and $B = B_{k^2} \cup B_k$, where $|A_k|, |B_k| = k$ and $|A_{k^2}|, |B_{k^2}| = k^2$, then construct a complete bipartite graph on $(A_{k^2}, B_k), (A_k, B_k), (A_k, B_{k^2})$, while having no edges between A_{k^2} and B_{k^2}. A vertex cover would be $A_k \cup B_k \cup \{s, t\}$, but we can achieve a max-flow in G of $\Omega(k^2)$.

Accordingly, the objective cannot be approximated using matching routines even in the static case. However, the solution can still be approximated using recent developments in flow algorithms [37, 38, 42]. Below we will show that these routines can be sped up on dynamic graphs using multiple layers of sparsification. Specifically, the cut sparsifiers from Section 1.6.4 allow us to dynamically maintain a $(1+\epsilon)$-approximation of the solution value, as well as some form of query access to the minimum cut, in $O(\text{poly}(\log n, \epsilon^{-1}))$ per update.

The section is organized as follows. Section 1.7.1 will give some of the high level ideas and critical observations on which our dynamic algorithm will hinge. Section 1.7.2 will present the dynamic algorithm for maintaining a $(2+\epsilon)$-approximate minimum $s-t$ cut, prove that the approximation factor is correct, and show that the dynamic update time is $O(\text{poly}(\log n, \epsilon^{-1}))$ if we can dynamically update all data structures necessary for the algorithm in $O(\text{poly}(\log n, \epsilon^{-1}))$ time. Finally, Section 1.7.3 will present all of the necessary data structures and show how we can dynamically maintain them in $O(\text{poly}(\log n, \epsilon^{-1}))$ time.

1.7.1 Key Observations and Definitions

Our starting point is the observation that a small solution value implies a small vertex cover.

Lemma 1.4.1. *The minimum vertex cover in G has size at most $OPT + 2$ where OPT is the size of the minimum $s-t$ cut in G.*

Proof. Denote the minimum vertex cover as MVC, and the minimum $s-t$ cut in G as $(S, \bar{S})$ where $S = \{s\} \cup A_s \cup B_s$ and $\bar{S} = \{t\} \cup A_t \cup B_t$. Hence, we must have $OPT \geq |A_t| + |B_s| + |E(A_s, B_t)|$ where $E(A_s, B_t)$ are all of the edges between A_s and B_t.

Let $V_A(A_s, B_t)$ denote all of the vertices in A that are incident to an edge in $E(A_s, B_t)$, so $|V_A(A_s, B_t)| \leq |E(A_s, B_t)|$. We know $G_{A,B}$ is bipartite, so $A_t \cup B_s \cup V_A(A_s, B_t)$ must be a vertex cover in $G_{A,B}$, which implies $|MVC| \leq OPT + 2$ by adding s and t to the cover. □

Our goal, for the rest of this section, is to show ways of reducing the graph onto a small vertex cover, while preserving the flow value. The first issue that we encounter is that the minimum vertex cover can also change during the updates. However, in our case, the low arboricity property of the sparsifier given in Corollary 1.6.2 gives a more direct way of obtaining a small cover:

Lemma 1.7.2. *For any tree T, the vertex cover of all vertices other than the leaves is within a 2-approximation of the minimum vertex cover.*

This is proven in Appendix 1.11. We suspect that this is a folklore result, but it was difficult to find a citation of it, as there exist far better algorithms for maintaining vertex covers on dynamic trees [77]. Since there are at most $O(\text{poly}(\log n, \epsilon^{-1}))$ trees, and the overall vertex cover needs to be at least the size of any cover in one of the trees, we can set the cover as the set of all non-leaf vertices in the trees.

Definition 1.7.3. Given a set of disjoint spanning forests $F = F_1 \cup \ldots \cup F_K$, we say that $VC = \bigcup_{i \in [K]} VC_i$ is a **branch vertex cover** of F, if each VC_i is the set of all vertices other than the leaves in F_i

Corollary 1.7.4. *For any graph $G = (V, E)$ and corresponding sparsified graph $\tilde{G} = F_1 \cup \ldots \cup F_K$. If VC is a **branch vertex cover** of $\tilde{G}$, then, VC is a $2K$-approximate vertex cover of $\tilde{G}$. Furthermore, any $x \in V \setminus VC$ has degree at most K in $\tilde{G}$*

Proof. Since the size of a minimum vertex cover in subgraph can only be smaller, we have

$$|MVC_{F_i}| \leq |MVC_G|.$$

Coupling this the choice of $|VC|$ gives $|VC_i| \leq 2|MVC_{F_i}|$, and summing over all K trees gives the bound. The bound on the degree of x follows from all leaves having degree 1. □

We will ensure that s and t are placed in the cover, and use X to denote the non-cover vertices. If we let the neighborhood of x be $N(x)$, its interaction with various partitions of S can be described as:

Definition 1.7.5. For a cut on VC, $S \subseteq VC$, and a non-cover vertex $x \in X$ with neighborhood $N(x)$, let

1. $w(x, S) \stackrel{\text{def}}{=} \sum_{u \in S \cap N(x)} w(x, u)$,

2. $w^{(x)}(S) \stackrel{\text{def}}{=} \min\{w(x,S), w(x, VC \setminus S)\}$.

Definition 1.7.6. Given a graph $G = (V, E)$ and some $\widehat{V} \subseteq V$ such that $\widehat{V}$ is a vertex cover of G, and $X = V \setminus \widehat{V}$

1. For any $S \subset V$, let $\Delta_G(S)$ be the weight of cut S on G

2. For any $S_{\widehat{V}} \subset \widehat{V}$, let the weight of a cut that is minimally extended from $S_{\widehat{V}}$ then be given by

$$\Delta_G(S_{\widehat{V}}) \stackrel{\text{def}}{=} \Delta_{G \setminus X}(S_{\widehat{V}}) + \sum_{x \in X} w^{(x)}(S_{\widehat{V}}),$$

Definition 1.7.7. Given $G = (V_G, E_G)$ and $H = (V_H, E_H)$ such that $V_H \subseteq V_G$ and $\widehat{V}$ is a vertex cover of both graphs

1. If $V_H = V_G$, then we say $H \approx_\epsilon G$ if for any $S \subset V_G$

$$(1-\epsilon)\Delta_H(S) \leq \Delta_G(S) \leq (1+\epsilon)\Delta_H(S)$$

2. If $V_H \subset V_G$, then we say $H \approx_\epsilon^{\widehat{V}} G$, if for any $S_{\widehat{V}} \subset \widehat{V}$

$$(1-\epsilon)\Delta_H(S_{\widehat{V}}) \leq \Delta_G(S_{\widehat{V}}) \leq (1+\epsilon)\Delta_H(S_{\widehat{V}})$$

Note that if some $x \in X$ has degree 1, it will always belong to the same side as its neighbor in a minimum $s-t$ cut; while if x is incident to two neighbors u and v, it will always go with the neighbor with smaller weight. That means that if $w(x,u) \leq w(x,v)$, then this is equivalent to an edge of weight $w(x,u)$ between u and v. This suggests that we can reduce the star out of x, N_x, to a set of edges on its neighborhood. We formalize the construction of this graph, K_x, as well as the resulting graph by removing all of X below:

Definition 1.7.8. Given a weighted graph $G = (V, E)$ and $w(u, v) \to \mathbb{R}_+$, and any S, let: K_x be the clique generated by running VERTEXELIMINATION: for any two neighbors u and v of x, the edge weight of $(u, v)_x$ is

$$\frac{w(x, v)w(x, u)}{\sum_{i \in N(x)} w(x, i)}.$$

For some vertex cover VC and independent set $X = V \setminus VC$, we let $G_{VC} = (G \setminus X) \cup \bigcup_{x \in X} K_x$

Note that we're using a subscript $_x$ to denote the origin of the edge. Specifically, an edge $e_x \in G_{VC}$ implies that $e_x \in K_x$, and an edge $e_\emptyset \in G_{VC}$ means it's from VC, i.e. $e_\emptyset \in G \setminus X$. Note that G_{VC} also defines a weight for each cut $S_{VC} \subset VC$, where $\Delta_{G_{VC}}(S_{VC})$. The crucial property of Definition 1.7.8 is that it preserves the values all cuts within a factor of 2. We prove the following in Appendix 1.11.

Theorem 1.7.9. *Given a weighted graph $G = (V, E)$ and $w(u, v) \to \mathbb{R}_+$, with some vertex cover VC and independent set $X = V \setminus VC$. For any $S_{VC} \subset VC$*

$$\frac{1}{2}\Delta_G(S_{VC}) \leq \Delta_{G_{VC}}(S_{VC}) \leq \Delta_G(S_{VC})$$

Lemma 1.7.10. *Given $G = (V, E)$ with all weights in $[\gamma, \gamma U]$, along with vertex cover VC and independent set X, such that any $x \in X$ has degree at most d. Then the weight of any edge in G_{VC} is in $[\gamma(dU)^{-1}, \gamma U]$*

1.7.2 Dynamic Algorithm for Maintaining a Minimum $s - t$ Cut on Bipartite Graphs

Our algorithm can then be viewed as dynamically maintaining this cover using two layers of dynamic graph sparsifiers intermixed with elimination routines. Its main steps are shown in Figure 1.5.

One issue with maintaining a cut is that its two sides could have size $O(n)$, which cannot be returned in amortized $O(\text{poly}(\log n, \epsilon^{-1}))$ time. Instead, we will maintain the cut

1. Dynamically maintain a sparsified G, which we will denote $\tilde{G}$
2. Dynamically maintain a *branch vertex cover*, VC, of $\tilde{G}$, where we ensure $s, t \in VC$
3. Dynamically maintain multi-graph $\tilde{G}_{VC}$
4. Dynamically maintain a sparsified $\tilde{G}_{VC}$, which we will denote as H with vertex set V
5. Every $\frac{\epsilon}{2}\Delta_H(\widehat{S}_{VC})$ dynamic steps, recompute $\widehat{S}_{VC} \subset VC$, an approximate minimum $s-t$ cut on H, ignoring all degree zero vertices

Figure 1.5: Dynamic $(2+\epsilon)$-approximate Minimum $s-t$ Cut

$\widehat{S}_{VC} \subset VC$ with $s \in \widehat{S}_{VC}$, and allow querying of any vertex. For a vertex $v \in VC$, return v is with s iff $v \in \widehat{S}_{VC}$, which takes $O(1)$ time. For a vertex $x \notin VC$, return that x is with s iff $w(x, \widehat{S}_{VC}) = w^{(x)}(\widehat{S}_{VC})$ in $\tilde{G}$, taking $O(\text{poly}(\log n, \epsilon^{-1}))$ time to compute $w(x, \widehat{S}_{VC})$ and $w(x, VC \setminus \widehat{S}_{VC})$. Specifically, the cut will be

$$\widehat{S} = \widehat{S}_{VC} \cup \{x \in V \setminus VC : w(x, \widehat{S}_{VC}) = w^{(x)}(\widehat{S}_{VC})\},$$

the extension of $\widehat{S}_{VC}$ on $\tilde{G}$ which allows for the $O(\text{poly}(\log n, \epsilon^{-1}))$ query computation by Corollary 1.6.2 and Corollary 1.7.4.

We first establish the quality of this cut on H that we maintain:

Theorem 1.7.11. *Computing a $(1+\widehat{\epsilon})$-approximate minimum $s-t$ cut in H as in Step 5 of Figure 1.5 takes $O(OPT \cdot \text{poly}(\log n, \epsilon^{-1}))$ time for $\widehat{\epsilon} = \frac{\epsilon}{O(1)}$, and cut $\widehat{S}_{VC} \subset VC$ can be extended to $\widehat{S}$ a $2(1+\widehat{\epsilon})^5$-approximate minimum $s-t$ cut in G with high probability*

Proof. $\tilde{G} = F_1 \cup \ldots \cup F_K$ for some $K = O(\text{poly}(\log n, \epsilon^{-1}))$ by Corollary 1.6.2, so from Lemma 1.4.1 and Corollary 1.7.4, we know $|VC| = O(OPT \cdot \text{poly}(\log n, \epsilon^{-1}))$. From Corollary 1.6.2, the weights of $\tilde{G}$ are in $[1, O(n)]$, and Lemma 1.7.10 implies that the weights of $\tilde{G}_{VC}$ are in $[O(n^{-1}\,\text{poly}(\log n, \epsilon^{-1}))^{-1}, O(n)]$. Further, each K_x of $\tilde{G}_{VC}$ has at most $K^2 = O(\text{poly}(\log n, \epsilon^{-1}))$ edges, so $\tilde{G}_{VC}$ has $O(n \cdot \text{poly}(\log n, \epsilon^{-1}))$ edges. Corollary 1.6.2

then tells us that H has $O(|VC| \cdot \text{poly}(\log n, \epsilon^{-1})) = O(OPT \cdot \text{poly}(\log n, \epsilon^{-1}))$ edges, and that we can find a $(1+\widehat{\epsilon})$ approximate minimum $s-t$ cut in H, $\widehat{S}_{VC}$ in $O(OPT \cdot \text{poly}(\log n, \epsilon^{-1}))$ time.

From Corollary 1.6.2, we assume that $H \approx_{\widehat{\epsilon}} \tilde{G}_{VC}$ and $G \approx_{\widehat{\epsilon}} \tilde{G}$ with high probability.

Suppose $\widehat{S}_{VC} \subset VC$ is returned as a $(1+\widehat{\epsilon})$-approximate minimum $s-t$ cut in H, and let

$$\widehat{S} = \widehat{S}_{VC} \cup \{x \in V \setminus VC : w(x, \widehat{S}_{VC}) = w^{(x)}(\widehat{S}_{VC})\}$$

be its extension onto $\tilde{G}$. The left-hand side of Theorem 1.7.9 implies

$$\Delta_{\tilde{G}}(\widehat{S}_{VC}) \leq 2\Delta_{\tilde{G}_{VC}}(\widehat{S}_{VC}),$$

which along with the approximations $G \approx_{\widehat{\epsilon}} \tilde{G}$ and $\tilde{G}_{VC} \approx_{\widehat{\epsilon}} H$ gives

$$\Delta_G(\widehat{S}) \leq (1+\widehat{\epsilon})\Delta_{\tilde{G}}(\widehat{S}) \leq 2(1+\widehat{\epsilon})\Delta_{\tilde{G}_{VC}}(\widehat{S}_{VC}) \leq 2(1+\widehat{\epsilon})^2\Delta_H(\widehat{S}_{VC}).$$

On the other hand, let $\overline{S} \subset V$ be the minimum $s-t$ cut in G, and $\overline{S}_{VC} \subset VC$ be its restriction to VC. Since right-hand side of Theorem 1.7.9 is over optimum choices of $V \setminus S_{VC}$, we have

$$\Delta_{\tilde{G}}(\overline{S}) \geq \Delta_{\tilde{G}}(\overline{S}_{VC}) \geq \Delta_{\tilde{G}_{VC}}(\overline{S}_{VC}),$$

which when combined with the approximations $G \approx_{\widehat{\epsilon}} \tilde{G}$ and $\tilde{G}_{VC} \approx_{\widehat{\epsilon}} H$ gives

$$\Delta_G(\overline{S}) \geq (1-\widehat{\epsilon})\Delta_{\tilde{G}}(\overline{S}) \geq (1-\widehat{\epsilon})\Delta_{\tilde{G}_{VC}}(\overline{S}_{VC}) \geq (1-\widehat{\epsilon})^2\Delta_H(\overline{S}_{VC}).$$

The result then follows from the near-optimality of $\widehat{S}_{VC}$ on H, $\Delta_H(\overline{S}_{VC}) \geq (1-\widehat{\epsilon})\Delta_H(\widehat{S}_{VC})$.

□

Corollary 1.7.12. *The dynamic algorithm maintains a $(2+\epsilon)$-approximate minimum $s-t$ cut in G, and will only compute an approximate minimum $s-t$ cut on H every $O(\epsilon OPT)$*

dynamic steps.

Proof. Choosing $\widehat{\epsilon} = \frac{\epsilon}{O(1)}$ in Theorem 1.7.11 can give a $(2+\frac{\epsilon}{2})$-approximate minimum $s-t$ cut in G. Borrowing notation from the proof of Theorem 1.7.11, an approximate minimum $s-t$ cut on H will be re-computed in $\frac{\epsilon}{2}\Delta_H(\widehat{S}_{VC})$ dynamic steps. $OPT = \Delta_G(\bar{S}) \leq \Delta_G(\widehat{S}) \leq 2(1+\widehat{\epsilon})^2\Delta_H(\widehat{S}_{VC})$, so $\Delta_H(\widehat{S}_{VC}) = O(OPT)$

□

1.7.3 Dynamically Updating Data Structures

As was shown in Corollary 1.7.12, the dynamic algorithm maintains a $(2+\epsilon)$-approximate minimum $s-t$ cut of G, an approximate minimum $s-t$ cut of H is computed every $O(\epsilon OPT)$, and that computation takes $O(OPT \cdot \text{poly}(\log n, \epsilon^{-1}))$ time from Theorem 1.7.11. Therefore, in order to establish that the amortized dynamic update time is $O(\text{poly}(\log n, \epsilon^{-1}))$, it suffices to show that all data structures can be maintained in $O(\text{poly}(\log n, \epsilon^{-1}))$ time per dynamic update, thereby finishing the proof of Theorem 1.7.1. As a result of Corollary 1.6.2, it suffices to show the following

Theorem 1.7.13. *For each addition/deletion of an edge in $\tilde{G}$, data structures for $\tilde{G}$, VC, $\tilde{G}_{VC}$, and H can be maintained in $O(\text{poly}(\log n, \epsilon^{-1}))$ time.*

Bounds on the dynamic update time of each data structure will all ultimately follow from the $O(\text{poly}(\log n, \epsilon^{-1}))$ degree bound for $\tilde{G}$ of all vertices not in the *branch vertex cover*, VC. This is a direct result of the $O(\text{poly}(\log n, \epsilon^{-1}))$ arboricity of $\tilde{G}$ from Corollary 1.6.2, and the properties of a *branch vertex cover* of $\tilde{G}$ in Corollary 1.7.4.

Data structure for $\tilde{G}$: A list of $O(\text{poly}(\log n, \epsilon^{-1}))$ spanning forests, which we will denote SPANNERS_G.

Data structure for adjacency lists of $\tilde{G}$:, We will denote it as $\text{ADJ-LIST}_{\tilde{G}}$, and it will have, for each vertex v, two lists LEAF_v and BRANCH_v:

INSERTVC(G, VC, v)

1. Delete all edges $e_v \in K_v$ in GRAPH$_{VC}$.
2. For all edges e adjacent to v in ADJ-LIST$_{\tilde{G}}$, insert $e_\emptyset$ into GRAPH$_{VC}$.

Figure 1.6: Moving a Vertex into VC

- The list LEAF$_v$ will have the adjacency list of v for each spanning forest in SPANNERS$_G$ in which v is a leaf.
- Similarly, the list BRANCH$_v$ will have the adjacency list of v edge for each spanning forest in SPANNERS$_G$ in which v is not a leaf.

Data structure for VC: We will denote it as VC$_{\tilde{G}}$, which will be a list of all vertices v whose list BRANCH$_v$ is non-empty.

Data structure for $\tilde{G}_{VC}$: We will denote it as GRAPH$_{VC}$, and it will contain an adjacency list, ADJ$_v$, for each vertex $v \in VC$. Assume that each ADJ$_v$ has a data structure such that deletion and insertion of any edge takes $O(\log n)$ time.

Data structure for H: We will denote it as SPARSE$_{VC}$, and it will be the sparsified multi-graph.

We first show that moving a vertex in / out of the vertex cover can be done in time $O(\text{poly}(\log n, \epsilon^{-1}))$, assuming that the degree of the vertex added/removed is small. Note that the small number of forests in $\tilde{G}$ and the choice of VC allow us to meet this requirement.

Lemma 1.7.14. *If v is not in* VC$_{\tilde{G}}$*, then running* INSERT$_{VC}(v)$ *on* GRAPH$_{VC}$*, using* ADJ-LIST$_{\tilde{G}}$*, will output* GRAPH$_{VC}$ *equivalent to $\tilde{G}_{VC \cup v}$ of* ADJ-LIST$_{\tilde{G}}$ *in time* $O(\text{poly}(\log n, \epsilon^{-1}))$.

REMOVEVC(G, VC, v)

1. For all edges e adjacent to v in ADJ-LIST$_{\tilde{G}}$, delete $e_\emptyset$ from GRAPH$_{VC}$.
2. Use all incident edges to compute K_v and insert all $e_v \in K_v$ into GRAPH$_{VC}$

Figure 1.7: Removing a Vertex from VC

Proof. Costs of the two steps are:

1. Delete all edges $e_v \in K_v$ in GRAPH$_{VC}$. This requires finding all incident vertices to v in LEAF$_v$ and BRANCH$_v$, which is at most $O(\text{poly}(\log n, \epsilon^{-1}))$ because BRANCH$_v$ is empty due to v not in VC$_{\tilde{G}}$. Every pair of vertices has a corresponding edge e_v in GRAPH$_{VC}$, so this takes $O(\text{poly}(\log n, \epsilon^{-1}))$ time.

2. There are at most $O(\text{poly}(\log n, \epsilon^{-1}))$ edges adjacent to v in ADJ-LIST$_{\tilde{G}}$, so adding all these edges into GRAPH$_{VC}$ takes $O(\text{poly}(\log n, \epsilon^{-1}))$ time.

If v is not in VC$_G$, then v must only be incident to VC in ADJ-LIST$_{\tilde{G}}$. Therefore in $\tilde{G}_{VC\cup v}$, v will only be incident to edges $e_\emptyset$ for each e incident to v in ADJ-LIST$_{\tilde{G}}$, and no edges e_v will be in $\tilde{G}_{VC\cup v}$. INSERT$_{VC}(v)$ will perform exactly these operations on GRAPH$_{VC}$.

□

Lemma 1.7.15. *If* BRANCH$_v$ *is empty, then running* REMOVE$_{VC}(v)$ *on* GRAPH$_{VC}$*, using* ADJ-LIST$_{\tilde{G}}$*, will output* GRAPH$_{VC}$ *equivalent to* $\tilde{G}_{VC\setminus v}$ *of* ADJ-LIST$_{\tilde{G}}$ *in time* $O(\text{poly}(\log n, \epsilon^{-1}))$.

Proof. Costs of the two steps are:

1. At most $O(\text{poly}(\log n, \epsilon^{-1}))$ edges are adjacent to v in ADJ-LIST$_{\tilde{G}}$, so deleting all these edges from GRAPH$_{VC}$ takes $O(\text{poly}(\log n, \epsilon^{-1}))$ time.

UPDATEADJ(G, VC, e)

1. If e has been added/deleted, then add/delete e from the adjacency list of u and v for F_i in ADJ-LIST$_{\tilde{G}}$, which will be denoted $L_{u,i}$ and $L_{v,i}$, respectively.
2. For u and v, if $L_{v,i}$ has at most one adjacent vertex, place it in LEAF$_v$, otherwise place it in BRANCH$_v$.
3. If the degree of u and v in F_i is zero before adding e, then place $L_{v,i}$ in BRANCH$_v$ and $L_{u,i}$ in BRANCH$_u$
4. For u and v, if degree of v is two before deleting e, check the other vertex incident to v, say it is w, and if w has degree one in F_i then move $L_{v,i}$ to BRANCH$_v$ and $L_{w,i}$ to BRANCH$_w$.
5. For u and v, if degree of v is one before adding e, check the other vertex incident to v, say it is w, and if w has degree one in F_i then move $L_{w,i}$ to LEAF$_w$.

Figure 1.8: Update ADJ-LIST$_{\tilde{G}}$

2. $v \notin VC$, so v has $O(\text{poly}(\log n, \epsilon^{-1}))$ neighbors, and using all incident edges to compute each $e_v \in K_v$ and insert e_v into GRAPH$_{VC}$ takes $O(\text{poly}(\log n, \epsilon^{-1}))$ time.

If BRANCH$_v$ is empty, then v must only be incident to VC in ADJ-LIST$_{\tilde{G}}$. Therefore in $\tilde{G}_{VC\setminus v}$, v will never be incident to any edges $e_\emptyset$, and for any of its neighbors w and z, $(w, z)_v$ will be in ADJ-LIST$_{\tilde{G}}$. INSERT$_{VC}(v)$ will perform exactly these operations on GRAPH$_{VC}$

□

We now consider updating ADJ-LIST$_{\tilde{G}}$ given the addition/deletion of some edge. This process is simple in terms of time complexity, but has a small wrinkle in maintaining the correct LEAF and BRANCH structure. Specifically, for each forest, we can consider all of the degree one vertices to be leaves, except for when there is a disjoint edge in the forest. Accordingly, steps 3, 4, and 5 of the algorithm in Figure 1.8 will take care of this edge case.

Lemma 1.7.16. UPDATEADJ(G, VC, e) *takes* $O(\log n)$ *time and all vertices* v *such that* $L_{v,i}$ *are in* BRANCH$_v$*, maintain a 2-approximate vertex cover of* F_i*.*

Proof. Finding the adjacency list of u and v for F_i in ADJ-LIST$_{\tilde{G}}$ takes $O(\log n)$ time. The rest of the steps all take $O(1)$ time, as they are just there to ensure we maintain the 2-approximate vertex cover of F_i.

For all trees, other than a single edge, it suffices to put all vertices with degree ≥ 2 in the vertex cover, and 2-approx tree theorem tells us that this is a 2-approximate vertex cover. Step 3 and 4 of Update ADJ-LIST$_{\tilde{G}}$ ensure that in the single edge case, $e = (u, v)$ that $L_{v,i}$ is in BRANCH$_v$ and $L_{u,i}$ is in BRANCH$_u$, which is still a 2-approximate vertex cover. Further, step 5 ensures that anytime an edge is added to a tree that just contains a single edge, all vertices of degree one have their adjacency list moved to the LEAF list. □

Full Dynamic Update Process

Finally, we consider the addition/deletion of an edge in SPANNERS$_G$. Specifically, let the edge $e = (u, v)$ be added/deleted from forest F_i. The above two operations allow us to reduce it to the simpler case of both u and v being in VC. The update process will occur as follows:

1. For u and v, if $v \notin$ VC$_{\tilde{G}}$, then run INSERTVC on GRAPH$_{VC}$, VC$_{\tilde{G}}$, and v
2. Update ADJ-LIST$_{\tilde{G}}$
3. If e was added/deleted from $\tilde{G}$, insert/delete edge $e_\emptyset$ from GRAPH$_{VC}$ and insert u and v into VC$_{\tilde{G}}$
4. For u and v, if BRANCH$_v$ is empty, then run REMOVEVC on GRAPH$_{VC}$, VC$_{\tilde{G}}$, and v, and delete v from VC$_{\tilde{G}}$

By Lemma 1.7.14, GRAPH$_{VC}$ is equivalent to $\tilde{G}_{VC\cup\{u,v\}}$ on updated ADJ-LIST$_{\tilde{G}}$ after step 3 because u and v are in VC. Similarly, the moving of u and v outside of VC ensures our final state is good.

Proof of Theorem 1.7.13 : The full update process for ADJ-LIST$_{\tilde{G}}$, VC$_{\tilde{G}}$, and GRAPH$_{VC}$ only calls INSERTVC, REMOVEVC, and UPDATEADJ a constant number of times. Therefore, by Lemma 1.7.14, Lemma 1.7.15, and Lemma 1.7.16 this process only takes time $O(\text{poly}(\log n, \epsilon^{-1}))$. This also implies that at most $O(\text{poly}(\log n, \epsilon^{-1}))$ edges can be added/deleted from $\tilde{G}_{VC}$, and by Corollary 1.6.2 maintaining H will take at most time $O(\text{poly}(\log n, \epsilon^{-1}))$.

1.8 Vertex Sampling in Bipartite Graphs

We now design an improved method for reducing a graph onto one whose vertex size is $O(|VC| \text{ poly}(\log n, \epsilon^{-1})) + |X|/2$. Instead of sampling edges of G_{VC}, it samples vertices in $X = V \setminus VC$ using G_{VC} as a guide. This question that we're addressing, and the vertex sampling scheme, is identical to the terminal cut sparsifier question addressed in [2]. In the next section we will apply this sampling scheme to obtain a vertex sparsification routine that will reduce onto a graph of size proportional to $O(|VC| \text{ poly}(\log n, \epsilon^{-1}))$ without losing a factor of 2 approximation.

We will reuse the notation from Section 1.7.1, and we encourage the reader to revisit the definitions in that subsection. For this section, we will exclusively be dealing with subsets of VC, and we will drop the VC subscript from each S_{VC}. So, formally our goal is to find H so that for all $S \subset VC$,

$$(1-\epsilon)\Delta_G(S) \leq \Delta_H(S) \leq (1-\epsilon)\Delta_G(S).$$

This sampling scheme allows us to keep expectation of the cuts on VC to be exactly the same, instead of having a factor 2 error from the conversion from G to G_{VC}. The connection to G_{VC} on the other hand allows us to bound the variance of this sampling process as before.

In our application of this sampling routine to vertex sparsification, we will consider sparsifying $G \setminus X$ separately, so for simplicity, we assume here that (VC, X) is a bipartition

and

$$G = \bigcup_{x \in X} N_x$$

Further, we first focus on the case where all vertices in X have degree d, and all edge weights in X are within a factor of U from each other. We will show reductions from general cases to ones meeting these assumptions in Subsection 1.9.1.

As before, let G_{VC} be the multigraph generated by the clique edges from Theorem 1.7.9:

$$G_{VC} = \bigcup_{x \in X} K_x.$$

Lemma 1.7.10 implies that the weights of every (multi) edge $e_x \in G_{VC}$ are within a factor of $O(U^2 d)$ from each other.

As mentioned, we ultimately want to obtain a vertex sparsification scheme that reduces to size $O(|VC| \operatorname{poly}(\log n, \epsilon^{-1}))$ for further application. As a result, instead of doing a direct union bound over all $2^{|VC|}$ cuts to get a size of $\operatorname{poly}(|VC|)$ as in [2], we need to invoke cut counting as with cut sparsifier constructions. This necessitates the use of objects similar to t-bundles to identify edges with small connectivity.

Our proof will use a similar structure to that of Fung et al. [64], particularly the cut-counting based analysis of cut sparsifiers. We will follow their definitions, which are in turn based on the definition of edge strength by Benczur and Karger [33].

Definition 1.8.1. In a graph G, an edge is e $k - heavy$ if the connectivity of its endpoints is at least k in G. Furthermore, for a cut S, its $k - projection$ is the set of $k - heavy$ edges in the edges cut, $\partial(S)$.

We will refer to edges that we cannot certify to be heavy as light. These edges are analogous to the bundle edges from the cut sparsifier routine from Section 1.6.4.

Before we continue, we remark that the definition of heavy/strong edges in [64, 33] is almost the opposite of definitions in spectral sparsification. In spectral sparsification, the

edges with high leverage scores are kept, and the low leverage score ones are sampled. This issue can also be reflected in the robustness of this definition in the presence of weights: a natural way of generalizing heaviness is to divide the connectivity of uv by the weight $w(u, v)$. This leads to a situation where halving the weight of an edge actually makes it heavier. In fact, these definitions of heaviness / strength are measuring the connectivity in the graph between the endpoints of e, instead of the strength of e itself. As our routines are in the cut-sparsification setting, we will use these definitions in this version in order to be consistent with previous works [64, 33], but may switch to a different set of notations in a future edit.

The main result of [64], when restricted to graphs with bounded edge weights, states that we can sample the $O(\log n\epsilon^{-2})$-heavy edges by a factor of 2. Our goal is to prove the analogous statement for sampling heavy vertices, which we define as follows:

Definition 1.8.2. A subset of X, X^{heavy} is a k-heavy subset if every pair of vertices u, v in some N_x for some $x \in X^{heavy}$ is k-connected in the graph

$$G_{VC}^{light} = \cup_{x \notin X^{heavy}} K_x.$$

We will show in Section 1.9, these heavy/light subsets can be found by taking pre-images of more restricted versions of t-bundles on G_{VC}. Our main structural result is that a heavy subset can be sampled uniformly while incurring ϵ-distortion.

Lemma 1.8.3. *Given a bipartite graph G between VC and X such that X has maximum degree d and all edge weights are in some range $[\gamma, U\gamma]$, with $U = O(\text{poly}(n))$ and any non-negative γ. For any ϵ, there is a parameter $t_{\min} = O(dU \log n\epsilon^{-2})$ such that if we're given a subset X^{light} of X so that $X^{heavy} = X \setminus X^{light}$ is $\gamma(dU)t$-heavy with $t \geq t_{\min}$ then the graph consisting of the light vertices and sampled heavy vertices,*

$$H = N(X^{light}) \cup \textsc{Sample}(G, VC, X^{heavy})$$

SAMPLE(G, VC, X^{heavy})
Input: Bipartite graph G with one bipartition VC, heavy subset X^{heavy} of the other bipartition.
Output: Bipartite graph H with bipartition (VC, XH).

1. Initialize $H \leftarrow \emptyset$, $XH = \emptyset$.
2. For every $x \in X^{heavy}$, flip fair coin with probability $1/2$, if returns heads:
   (a) $H \leftarrow H + 2N_x$.
   (b) $XH \leftarrow XH \cup \{x\}$
3. Return (H, XH).

Figure 1.9: Sampling Heavy Vertices

meets the condition:

$$|\Delta_G(S) - \Delta_H(S)| \leq \epsilon \Delta_G(S)$$

for all subsets $S \subseteq VC$ w.h.p. Here the constants in $t_{\min}$ depends on the failure probability in the w.h.p.

The cut-counting proof of cut-sparsifiers from [64] essentially performs a union bound over distinct sets of k-heavy projections over all cuts. We will perform the same here, but over distinct partitions of N_x over all x in X^{heavy}. We can first define the partition of a single vertex by a cut $S \subseteq VC$ as:

$$N_x(S) = \{S \cap N(x), N(x) \setminus S\}.$$

Then we can define an equivalence relation on cuts as:

Definition 1.8.4. $S_1 \equiv_G S_2$ if for any $x \in X^{heavy}$, $N_x(S_1) = N_x(S_2)$

Note that this equivalence ignores the presence of edges in X^{light}. So we need to further take representatives of each equivalence class:

Definition 1.8.5. Define $\mathcal{S}^{rep}$ to be the set of subsets $S \subset VC$ such that

1. For every $S \in \mathcal{S}^{rep}$, there is some $x \in X^{heavy}$ s.t. $N_x(S) \neq \{N_x, \emptyset\}$, i.e. N_x is not entirely on one side of the cut.

2. For any $S_1, S_2 \in \mathcal{S}^{rep}$, $S_1 \not\equiv_G S_2$

3. For any $S \subset VC$ such that $S \notin \mathcal{S}^{rep}$, there exists $\overline{S} \in \mathcal{S}^{rep}$ such that

   - $S \equiv_G \overline{S}$, and
   - $\Delta_G(\overline{S}) \leq \Delta_G(S)$.

An immediate consequence of condition 1 is that for any $S \in \mathcal{S}^{rep}$ we have $\Delta_G(S) > \gamma t(dU)^{-1}$. This set plays the same role as the unique k-projections in cut sparsifiers.

Lemma 1.8.6. *Let H be obtained from G by sampling on X^{heavy}, then for any element of $\mathcal{S}^{rep}$, $\overline{S}$ we have:*

$$\mathbb{P}_H\left[\bigcup_{S, S\equiv_G\overline{S}} |\Delta_G(S) - \Delta_H(S)| > \epsilon\Delta_G(S)\right] = \mathbb{P}_H\left[\left|\Delta_G(\overline{S}) - \Delta_H(\overline{S})\right| > \epsilon\Delta_G(\overline{S})\right].$$

Proof. Let $G_{sample} = G \setminus \bigcup_{x\in X^{light}} N_x$ and $H_{sample} = H \setminus \bigcup_{x\in X^{light}} N_x$ be the graphs being sampled.

By construction of H, for any $S \subset VC$,

$$|\Delta_G(S) - \Delta_H(S)| = \left|\Delta_{G_{sample}}(S) - \Delta_{H_{sample}}(S)\right|.$$

By construction of our equivalence relation, if $S \equiv_G \overline{S}$,

$$\left|\Delta_{G_{sample}}(S) - \Delta_{H_{sample}}(S)\right| = \left|\Delta_{G_{sample}}(\overline{S}) - \Delta_{H_{sample}}(\overline{S})\right|.$$

due to them having the same part that's not in H. Therefore, the failure probability is limited by the element in the equivalence class with the smallest $\Delta_G(S)$, i.e. $\overline{S}$. □

Corollary 1.8.7.

$$\mathbb{P}_H\left[\bigcup_{S\subset VC} |\Delta_G(S) - \Delta_H(S)| > \epsilon\Delta_G(S)\right] = \mathbb{P}_H\left[\bigcup_{S\in\mathcal{S}^{rep}} |\Delta_G(S) - \Delta_H(S)| > \epsilon\Delta_G(S)\right]$$

The key observation is that the sizes of subsets of $\mathcal{S}^{rep}$ of certain sizes can be bounded using cut-counting on G_{VC}. For any $S \subset VC$, define

$$K_x(S) = E_{K_x}(S \cap N(x), N(x) \setminus S),$$

which are the edges in K_x crossing S. Similar to $S_1 \equiv_G S_2$, we can define $S_1 \equiv_{G_{VC}} S_2$ if for any $x \in X^{heavy}$, $K_x(S_1) = K_x(S_2)$

Lemma 1.8.8. *For any $S_1, S_2 \subseteq VC$, $N_x(S_1) = N_x(S_2)$ iff $K_x(S_1) = K_x(S_2)$. Therefore $S_1, S_2 \subseteq VC$, $S_1 \equiv_G S_2$ iff $S_1 \equiv_{G_{VC}} S_2$.*

Proof. We construct K_x as a clique, so $K_x(S_1) = K_x(S_2)$ iff $S_1 \cap N(x) = S_2 \cap N(x)$ or $S_1 \cap N(x) = N(x) \setminus S_2$ □

Lemma 1.8.9. *$|\{S \in \mathcal{S}^{rep} | \Delta_{G_{VC}}(S) \leq K\}|$ is less than or equal to the number of distinct $\gamma(dU)^{-1}t$-projections in cuts of weight at most K*

Proof. Lemma 1.8.8 gives that $\mathcal{S}^{rep}$ has the following properties for G_{VC}

1. For every $S \in \mathcal{S}^{rep}$, there is some $x \in X^{heavy}$ s.t. $K_x(S) \neq \emptyset$.
2. For any $S_1, S_2 \in \mathcal{S}^{rep}$, $S_1 \not\equiv_{G_{VC}} S_2$

For any $S \in \mathcal{S}^{rep}$, let $E_{heavy}(S)$ denote all the $\gamma(dU)^{-1}t$-heavy edges crossing S in G_{VC}. The property above gives:

$$\bigcup_{x\in X^{heavy}} K_x(S)$$

is a non-empty subset of $E_{heavy}(S)$, and

$$\bigcup_{x\in X^{heavy}} K_x(S_1) \neq \bigcup_{x\in X^{heavy}} K_x(S_2) \qquad \forall S_1, S_2 \in \mathcal{S}^{rep}.$$

Therefore, each $S \in \mathcal{S}^{rep}$ such that $\Delta_{G_{VC}}(S) \leq K$, must be a distinct $\gamma(dU)^{-1}t - projection$ of weight at most K

□

It remains to combine this correspondence with cut counting to show the overall success probability of the vertex sampling routine.

Proving this requires using Chernoff bounds. The bound that we will use is below, it can be viewed as a scalar version of Theorem 1 of [57].

Lemma 1.8.10. *Let $Y_1 \dots Y_n$ be random variables s.t.*

1. $0 \leq Y_i \leq 1$.
2. $\mu_i = \mathbb{E}_{Y_i}[Y_i]$
3. $\mu = \sum_i \mu_i$

Then for any $\epsilon \geq 0$

$$\mathbb{P}_{Y_1 \dots Y_n}\left[\sum_i Y_i > (1+\epsilon)\mu\right] \leq \exp\left(-\frac{\epsilon^2\mu}{2}\right).$$

$$\mathbb{P}_{Y_1 \dots Y_n}\left[\sum_i Y_i < (1-\epsilon)\mu\right] \leq \exp\left(-\frac{\epsilon^2\mu}{2}\right).$$

This bound can be invoked in our setting on a single cut S as follows:

Lemma 1.8.11. *For each cut S, we have*

$$\mathbb{P}_H\left[|\Delta_H(S) - \Delta_G(S)| > \epsilon\Delta_G(S)\right] \leq 2\exp\left(-\frac{\epsilon^2\Delta_G(S)}{4\gamma}\right).$$

Proof. Let

$$w^{\max}(S) = \max_{x \in X}\{w^{(x)}(S)\}.$$

We will only consider $S \in \mathcal{S}^{rep}$, so we know $w^{\max}(S) > 0$, which implies $w^{\max}(S) \geq \gamma$. For each $S \subseteq \mathcal{S}^{rep}$ and for all $x \in X$, let $Y_x(S)$ be the random variable such that either

1. $Y_x(S) = \frac{w^{(x)}(S)}{2w^{\max}(S)}$ if $x \in X^{light}$
2. $Y_x(S)$ equals $\frac{w^{(x)}(S)}{w^{\max}(S)}$ w.p. $1/2$, and 0 w.p. $1/2$.

Accordingly, we have $\sum_{x \in X} \mathbb{E}_{Y_x(S)}\left[Y_x(S)\right] = \frac{1}{2w^{\max}(S)} \sum_{x \in X} w^{(x)}(S) = \frac{1}{2w^{\max}(S)} \Delta_G(S)$. The bound then follows from invoking Lemma 1.8.10. □

Proof. (Of Lemma 1.8.3)

Let $\Delta_{G_{VC}}(S)$ be the weight of cutting $S \subset VC$ in G_{VC}. From Theorem 1.7.9 for any $S \in \mathcal{S}^{rep}$ that $\Delta_G(S) \geq \Delta_{G_{VC}}(S)$. Therefore,

$$\sum_{S \subseteq \mathcal{S}^{rep}} 2\exp\left(-\frac{\epsilon^2 \Delta_G(S)}{4\gamma}\right) \leq \sum_{S \subseteq \mathcal{S}^{rep}} 2\exp\left(-\frac{\epsilon^2 \Delta_{G_{VC}}(S)}{4\gamma}\right)$$

The main cut-counting bound follows from Theorem 1.6 [64] on multi-graphs, and by our construction of $\mathcal{S}^{rep}$ gives:

$$|\{S \in \mathcal{S}^{rep} | \Delta_{G_{VC}}(S) \leq K\}| \leq \begin{cases} n^{2KdU(\gamma t)^{-1}} & \text{if } K \geq \gamma(dU)^{-1}t, \\ 0 & \text{otherwise.} \end{cases}$$

Each vertex adds weight at most γdU for any cut, so we can upper bound K by $n^2\gamma U$ because $d \leq n$. Invoking cut counting for intervals of length γ from $K \geq \gamma(dU)^{-1}t$ to $K \leq n^2\gamma U$ allows us to bound the overall failure probability by:

$$\leq \sum_{i=(dU)^{-1}t}^{n^2U} \left(\sum_{\substack{S\in\mathcal{S}^{rep} \\ \gamma i\leq\Delta_{G_{VC}}(S)\leq\gamma(i+1)}} 2\exp\left(-\frac{\epsilon^2\Delta_{G_{VC}}(S)}{4\gamma}\right)\right)$$

$$\leq \sum_{i=(dU)^{-1}t}^{n^2U} 2n^{2(i+1)dUt^{-1}}\exp\left(-\frac{\epsilon^2 i}{4}\right) = \sum_{i=(dU)^{-1}t}^{n^2U} 2n^{2(i+1)dUt^{-1}-\frac{\epsilon^2 i}{4\log n}}. \quad (1.1)$$

Note that we're free to choose t, and it can be checked that for $U \leq n^{c_1}$, setting $t \geq (28+4c_1)c_2 dU\log n\epsilon^{-2}$ bounds this by n^{-c_2} for any $c_2 \geq 1$. Note that if U is larger than $O(\text{poly}(n))$, we could set $t = O(dU^2\log n\epsilon^{-2})$ and still achieve w.h.p., but for our practical purposes assuming $U = O(\text{poly}(n))$ is more than sufficient because U will always be $O(\text{poly}(\log n, \epsilon^{-1}))$.

□

1.9 Maintaining $(1+\epsilon)$-Approximate Undirected Bipartite Min-Cut

In this section, we will again consider the bipartite minimum $s-t$ cut problem of Section 1.7, and will improve the approximation guarantee to $(1+\epsilon)$. This improvement will require many of the techniques from Section 1.7, but we will bypass the loss of a factor 2 approximation by utilizing the vertex sampling scheme presented in Section 1.8. A high level overview of these techniques is in Section 1.4.3. The dynamic algorithm given in this section will rely heavily on the definitions and observations of Subsection 1.7.1, which we encourage the reader to revisit.

Lemma 1.8.3, along with the framework from Section 1.7 allow us sample a large set of vertices if the optimal minimum $s-t$ cut is small, and will guarantee that the sampled vertices have $O(\text{poly}(\log n, \epsilon^{-1}))$ degree. However, Lemma 1.8.3 as stated require incident edges of all sampled vertices to have weight within factor $O(\text{poly}(\log n, \epsilon^{-1}))$ of one another. In this section, we integrate this subroutine into the data structure framework, leading to our

main result for approximating undirected bipartite maximum flows:

Theorem 1.2.3. *Given a dynamically changing unweighted, undirected, bipartite graph $G = (A, B, E)$ with demand -1 on every vertex in A and demand 1 on every vertex in B, we can maintain a $(1-\epsilon)$-approximation to the value of the maximum flow, as well as query access to the associated approximate minimum cut, with amortized update time $\text{poly}(\log n, \epsilon^{-1})$ per edge insertion / deletion.*

Section 1.9.1 will show how the vertex sampling scheme given in Section 1.8 can be iteratively applied, reducing to a graph with $O(|VC| \text{poly}(\log n, \epsilon^{-1}))$ vertices and $O(|VC| \text{poly}(\log n, \epsilon^{-1}))$ edges. This section will first present the full vertex sparsification scheme, and then examine the two primary components of this scheme. Section 1.9.1 will show how we can pre-process a graph to ensure that all edge weights of each sampled vertex are close to each other, which will be necessary for bucketing sampled vertices. Section 1.9.1 will utilize these bounded properties and the vertex sampling of Section 1.8 to give a vertex sparsification scheme for each bucket, culminating in a proof of correctness for the full scheme in terms of approximation guarantees and bounds on the number of edges and vertices. Section 1.9.1 will extend vertex sparsification to general graphs without bounds on degree for the static case, proving Corollary 1.4.2.

Section 1.9.2 will then use this vertex sparsification scheme along with many of the components from Section 1.7 to give a fully dynamic algorithm for maintaining a minimum $s-t$ cut on a bipartite graph. The correctness of this algorithm will follow from the correctness of the dynamic algorithm in Section 1.7 and the correctness of vertex sparsification. Accordingly, it will then only be necessary to establish that we can dynamically update all necessary data structures in $O(\text{poly}(\log n, \epsilon^{-1}))$ time.

1.9.1 Vertex Sparsification in Quasi-Bipartite Graphs

The general framework of the routine is shown in Figure 1.10.

VERTEXSPARSIFY(G, VC, XG, d, ϵ)
Input: Graph G with vertex cover VC and $XG = V \setminus VC$, such that the degree of each vertex in XG is bounded by d.

1. Build $\widehat{G}$ on the same vertex set as G s.t. $G \approx_{\epsilon/2} \widehat{G}$ and for each x in $X\widehat{G}$, the weights are within a factor of $O(d/\epsilon)$ of each other.
2. Bucket $\widehat{G}$ by maximum edge weights in each N_x into $\widehat{G}_1 \ldots \widehat{G}_L$, along with $\widehat{G} \setminus XG$
3. Set $t = O(d^2 \log^3 n\epsilon^{-3})$, initialize $H = (VC, \emptyset)$.
4. With error $\epsilon/2$, sparsify $\widehat{G} \setminus XG$ and BOUNDEDVERTEXSPARSIFY each $\widehat{G}_i$, giving H_i
5. Return the union of each sparsified graph, $H = H \setminus XG \cup H_1 \cup \ldots \cup H_L$.

Figure 1.10: Vertex Sampling in G

Theorem 1.9.1. *Given any graph G, vertex cover VC and $XG = V \setminus VC$, such that the degree of each vertex in XG is bounded by d, with weights in $[\gamma, O(\gamma W)]$ where $\log W = O(poly(\log n))$, and error ϵ. Then there is a $t = O(d^2 \log^3 n\epsilon^{-3})$ whereby* VERTEXSPARSIFY(G, VC, XG, d, ϵ) *returns H s.t. w.h.p.*

1. *$H \setminus XG$ is a multi-graph on VC with $O(|VC| \operatorname{poly}(\log n, \epsilon^{-1}))$ edges, and each H_i is a bipartition with VC on one side, and at most $O(|VC|t \log n)$ vertices of XG on the other.*
2. *$H \approx_\epsilon^{VC} G$.*
3. *All edge weights of H are in $[\gamma, O(\gamma n W)]$*

Here the constant in front of t depends on W as well as in the w.h.p. condition.

A proof of Theorem 1.9.1 will be given at the end of Section 1.9.1.

Reduction to Bounded Weight Case

The idea here will be to look at each N_x and move the low weight edges into $G \setminus X$, thereby ensuring that the remaining edges in N_x have weight within a $O(\operatorname{poly}(\log n, \epsilon^{-1}))$ factor.

VERTEXBUCKETING(G, VC, XG, d, ϵ)
Input: Bipartite graph G with bipartition (VC, XG) s.t. the degree of each vertex in XG is bounded by d.

1. Initialize $\widehat{G} \setminus X = G \setminus X$, and $\widehat{G}_i = (VC, \emptyset)$ for $i = 1 \dots L$ with $L = O(\log W)$
2. For each $x \in XG$
   (a) Let (x, u) be the edge with maximum weight in N_x, where $w(x, u) \in [\gamma 2^{i-1}, \gamma 2^i]$
   (b) For each $(x, v) \in N_x$, if $w(x, v) < \frac{\epsilon}{d} w(x, u)$, then put $(u, v)_x$ in $\widehat{G} \setminus X$. Otherwise, put (x, v) in $\widehat{G}_i$
3. Return the multi-graph $\widehat{G} \setminus X$, and graphs $\widehat{G}_1 \dots \widehat{G}_L$

Figure 1.11: Vertex Bucketing in G

This will create a multi-graph in $G \setminus X$, where will use the normal notation $(u, v)_x$ to denote an edge added by N_x.

Theorem 1.9.2. *Given G with bipartition (VC, XG) with weights in $[\gamma, \gamma W]$, such that the degree of each vertex in XG is bounded by d, for any ϵ,* VERTEXBUCKETING(G, VC, XG, d, ϵ) *will return $\widehat{G} = \widehat{G} \setminus X \cup \widehat{G}_1 \cup \ldots \cup \widehat{G}_L$ such that*

1. *$G \approx_\epsilon \widehat{G}$*
2. *For each $\widehat{G}_i$, the weights of $\widehat{G}_i$ are in $[\gamma, 2\gamma d \epsilon^{-1}]$ for some γ*
3. *Any edge $e_\emptyset \in \widehat{G}$ must be in $\widehat{G} \setminus X$*
4. *If $x \in X$ has non-zero degree in $\widehat{G}_i$, then x has zero degree in $\widehat{G} \setminus \widehat{G}_i$, and the degree of x in $\widehat{G}_i$ is bounded by d*

Proof. Items 2, 3, and 4 follow from construction, and because XG is an independent set in G, we can conclude that $G \approx_\epsilon \widehat{G}$ from Lemma 1.9.3 and Lemma 1.9.4 below.

□

Lemma 1.9.3. *Consider a graph on three vertices, x, u, and v with edges between xu and xv. If $w(x,v) \leq \epsilon w(x,u)$, then the graph with edges xu with weight $w(x,u)$ and uv with weight $w(x,v)$ is an ϵ-approximation on all cuts.*

Proof. The only interesting cuts are singletons:

1. Removing v has $w(x,v)$ before and after.

2. Removing x has $w(x,u) + w(x,v)$ before, and $w(x,u)$ after, a factor of ϵ difference since
$$w(x,u) + w(x,v) \leq (1+\epsilon)w(x,u).$$

3. Removing u has $w(x,u)$ before, and $w(x,u) + w(x,v)$ after, same as above.

□

Invoking this repeatedly on small stars gives:

Lemma 1.9.4. *A star x with degree d can be reduced to one whose maximum and minimum weights is within a factor of $O(d\epsilon^{-1})$ while only distorting cuts by a factor of $1+\epsilon$.*

Proof. Let the neighbors of x be $v_1 \dots v_d$ s.t. $w(x,v_1) \geq w(x,v_2) \geq \dots \geq w(x,v_d)$. Suppose $w(x,v_i) < \epsilon/d w(x,v_1)$, then applying Lemma 1.9.3 gives a multiplicative error of $1+\epsilon/d$. Applying this at most d times gives the approximation ratio, and moves all the light edges onto v_1. □

Bounded Weight Vertex Sparsification

The bucketing of vertices in the independent set ensures that all the weights in each bucket are within a factor $O(d/\epsilon)$, which will allow us to iteratively reduce the number of vertices by applying the SAMPLE algorithm given in Section 1.8 $O(\log n)$ times. Note that our SAMPLE algorithm doubles the weights of each sampled star, so N_x^i will denote the star x in ith iteration graph G_i with updated weights for that graph.

BOUNDEDVERTEXSPARSIFY(G, VC, XG, t)
Input: Bipartite graph G with bipartition (VC, XG)

1. Initialize $G_0 \leftarrow G$, $XG_0 \leftarrow XG$, and $H \leftarrow \emptyset$
2. For each $i = 0$ to $l - 1$
   (a) Compute a t-bundle vertex set $XG_i^{light} \subseteq XG_i$ of G_i
   (b) $(G_{i+1}, XG_{i+1}) \leftarrow Sample(G_i, VC, XG_i, XG_i^{light})$
   (c) Add $\bigcup_{x \in XG_i^{light}} N_x^i$ to H
3. Return $H = H \cup G_l$

Figure 1.12: Bounded Weight Vertex Sparsification in G

Theorem 1.9.5. *Given a bipartite graph G with bipartition (VC, XG), and weights in $[\gamma, U\gamma]$ where $U = O(poly(n))$, with degree of $x \in XG$ bounded by d, and error ϵ. Then there is a $t = O(dU \log^3 n\epsilon^{-2})$ whereby* BOUNDEDVERTEXSPARSIFY(G, VC, XG, t) *returns H, s.t. w.h.p.*

1. *H is a bipartition with VC on one side and at most $O(|VC|t \log n)$ vertices on the other*
2. *$H \approx_\epsilon^{VC} G$*

Proof. **(1)**: Set $l = O(\log n)$ and note that $|XG| \leq n$, so G_l is unlikely to have many remaining vertices after sampling $O(\log n)$ times by a standard argument using concentration bounds. Then, Lemma 1.9.8 will show $|XG_i^{light}| \leq t|VC|$ for all i, giving the desired size. **(2)**: By construction of BOUNDEDVERTEXSPARSIFY(G, VC, XG), the weights of each G_i are in $[2^i\gamma, 2^i\gamma U]$. We will show in Lemma 1.9.8 that for each G_i and XG_i, we can find a t-bundle vertex set XG_i^{light} of XG_i, such that $XG_i^{heavy} = XG_i \setminus XG_i^{light}$ is a $2^i\gamma(dU)^{-1}t - heavy$ vertex subset. Assuming that this is the case, from Lemma 1.8.3, if

we set $\widehat{\epsilon} = \frac{\epsilon}{l}$, then with high probability

$$G_i \approx_{\widehat{\epsilon}}^{VC} G_{i+1} \cup \bigcup_{x \in XG_i^{light}} N_x^i$$

By construction, for all $j < i$, $XG_j^{light} \cap XG_i = \emptyset$, so adding each $\bigcup_{x \in XG_j^{light}} N_x^j$ to both sides will still preserve the relation above. Applying this argument inductively and using $\widehat{\epsilon} = \frac{\epsilon}{l}$ gives $H \approx_{\epsilon}^{VC} G$ with high probability.

□

In order to complete the proof of Theorem 1.9.5, it is now necessary to show that for each G_i and XG_i, we can construct XG_i^{light} such that $XG_i \setminus XG_i^{light}$ is a $2^i\gamma(dU)^{-1}t - heavy$ subset of XG_i. The idea will simply be to construct XG_i^{light} from t disjoint spanning forests in G_{VC}^i with some additional properties that will allow $O(\text{poly}(\log n, \epsilon^{-1}))$ dynamic maintenance in the following subsection.

Definition 1.9.6. Given G with vertex bipartition (VC, X), we say that $F = F_1 \cup \ldots \cup F_t$ is a t-clique forest if

1. Each F_i is a forest of G_{VC} and all are disjoint.
2. For any $x \in X$, at most one edge $e_x \in K_x$ is in F.
3. For all $x \in X$ such that $F \cap K_x = \emptyset$, for any $e_x = (u, v)_x \in K_x$, u and v are connected in all F_i

Lemma 1.9.7. *Given G with vertex bipartition (VC, X) such that all $x \in X$ have maximum degree d, weights in $[\gamma, \gamma U]$ and a t-clique forest F, if $X^{light} = \{x \in X | F \cap K_x = \emptyset\}$, then $X^{heavy} = X \setminus X^{light}$ is an $\gamma(dU)^{-1}t - heavy$ subset of X*

Proof. For some $(u, v)_x \in K_x$ with $x \in X^{heavy}$, suppose $(u, v)_x$ is in a cut $S_{VC} \subset VC$ such that $\Delta_{G_{VC}}(S_{VC}) < \gamma(dU)^{-1}t$. From Lemma 1.7.10, all edges in G_{VC} have weight at least

LIGHTVERTICES(G_i, VC, XG_i)
Input: Bipartite graph G_i with bipartition (VC, XG_i)

1. Initialize $XG_i^{light} \leftarrow \emptyset$ and $F_i = \bigcup_{j\in[t]} F_{i,j}$ with $F_{i,j} \leftarrow \emptyset$ for all j
2. For each $j = 1$ to t
   (a) While some edge $e_x \in G_{VC}^i$ can be added to forest $F_{i,j}$
   (b) Place e_x in $F_{i,j}$, place x in XG_i^{light}, and remove K_x from G_{VC}^i
3. Return XG_i^{light}

Figure 1.13: Light Vertex Set of XG

$\gamma(dU)^{-1}$. Therefore, there must exist some F_j such that u and v are not connected, giving a contradiction.

□

Note that after the algorithm terminates $G_{VC}^i = \bigcup_{x\in XG_i^{heavy}} K_x$, which will be necessary for the dynamic maintenance. The following lemma follows by construction and the fact that each forest has at most $|VC| - 1$ edges.

Lemma 1.9.8. *F_i is a t-clique forest of G_{VC}^i, $|XG_i^{light}| \leq t|VC|$, and XG_i^{heavy} is a $2^i\gamma(dU)^{-1}t - heavy$ subset of XG_i*

Proof of Theorem 1.9.1 **(1)** The first part follows from Theorem 1.6.1 and the second part follows from Theorem 1.9.5

(2) Property (1) of Theorem 1.9.2 gives us $G \approx_{\epsilon/2} \widehat{G}$ with $U = 4d\epsilon^{-1}$ for each $\widehat{G}_i$ from property (2). Then, property (3) implies that each $\widehat{G}_i$ is bipartite, and property (4) implies that each vertex in $X\widehat{G}_i$ is bounded by d. We can then apply Theorem 1.9.5 to each $\widehat{G}_i$, with $U = 4d\epsilon^{-1}$ to get $\widehat{G}_i \approx_{\epsilon/2}^{VC} H_i$ with high probability. Note that we are implicitly assuming $U = O(poly(n))$, aka $\epsilon^{-1} = O(poly(n))$. As was discussed at the end of Section 1.8, we could avoid this assumption by adding an extra ϵ^{-1} factor to the t-bundle,

but any $\epsilon^{-1} = \omega(poly(n))$ loses any practical value. $L = O(\log W) = O(poly(\log n))$ by assumption, and property (4) of Theorem 1.9.2 ensures that a vertex is only sampled in one $\widehat{G}_i$, so taking the union over $O(poly(\log n))$ buckets preserves $\widehat{G} \approx^{VC}_{\epsilon/2} H$ w.h.p. for sufficient constants in t. $G \approx_{\epsilon/2} \widehat{G}$ is a stronger statement than $G \approx^{VC}_{\epsilon/2} \widehat{G}$, implying $G \approx^{VC}_{\epsilon} H$

(3) Edge weights are only changed in SAMPLE where they are either doubled or left alone. VERTEXSPARSIFY calls SAMPLE at most $O(\log n)$ times for each bucket of $\widehat{G}$, giving the appropriate bound.

Improved Static Algorithm for General Graphs

Composing this routine $O(\log n)$ times, along with spectral sparsifiers, leads to a static routine:

Corollary 1.4.2. *Given any graph $G = (V, E)$, and a vertex cover VC of G, where $X = V \setminus VC$, with error ϵ, we can build an ϵ-approximate terminal-cut-sparsifier H with $O(|VC| \operatorname{poly}(\log n, \epsilon^{-1}))$ vertices in $O(m \cdot \operatorname{poly}(\log n, \epsilon^{-1}))$ work.*

Now that we have sufficient notation in place, by $terminal - cut - sparsifier$, we mean that $G \approx^{VC}_{\epsilon} H$ with high probability. Note that this is almost equivalent to Theorem 1.9.1, but we make no assumptions on the degree of vertices in X. Also, we will specify $\operatorname{poly}(\log^n, \epsilon^{-1})$ as $\log^{18} n\epsilon^{-7}$.

Proof. Consider running the following routine iteratively:

1. Sparsify G with error $\widehat{\epsilon} = \frac{\epsilon}{O(\log n)}$ and output $\tilde{G}$
2. Find the bipartite subgraph $\widehat{G}$ containing VC and vertices $X\widehat{G} \subseteq X$ whose degree are less than $O(\log^2 n\epsilon^{-2})$. Run VERTEXSPARSIFY on $\widehat{G}$, VC, $X\widehat{G}$, with $d = O(\log^2 n\epsilon^{-2})$ and with error $\widehat{\epsilon} = \frac{\epsilon}{O(\log n)}$, returning $\widehat{H}$
3. $G \leftarrow \tilde{G} \setminus \widehat{G}$ and $H \leftarrow H \cup \widehat{H}$

If at any point, we have $|X| < |VC| \log^{17} n\epsilon^{-7}$, then return $H \cup G$.

From [55], and the number of edges in $\tilde{G}$ is $O(n \log n\widehat{\epsilon}^{-2})$ with high probability. Therefore, at least half of $|X|$ have degree less than $O(\log^2 n\widehat{\epsilon}^{-2})$ because otherwise the number of edges in $\tilde{G}$ would be $O(|X| \log^2 n\widehat{\epsilon}^{-2}) = O(n \log^2 n\widehat{\epsilon}^{-2})$ by the assumption $|X| \geq |VC|$. This eliminates half the vertices in X with high probability for every run of the routine, so the process can continue at most $O(\log n)$ times. From Theorem 1.9.1 each bucket of $\widehat{H}$ will have at most $O(|VC|t \log n)$ vertices with $t = O(d^2 \log^3 n\widehat{\epsilon}^{-3})$ and $d = O(\log^2 n\widehat{\epsilon}^{-2})$, giving $O(|VC| \log^{15} n\epsilon^{-7})$. We run sparsification on G and VERTEXSPARSIFY on $\widehat{G}$ $O(\log n)$ times, so from the guarantees of Theorem 1.6.1 and property (3) of Theorem 1.9.1, the weights are within a factor $O(n^{O(\log n)})$. Therefore, there are at most $O(\log^2 n)$ buckets of $\widehat{H}$, and at most $O(|VC| \log^{17} n\epsilon^{-7})$ vertices which has the appropriate size requirement.

Sparsification gives $G \approx_{\widehat{\epsilon}} \tilde{G}$ with high probability, which is a stronger statement than $G \approx^{VC}_{\widehat{\epsilon}} \tilde{G}$. Theorem 1.9.1, which is still applicable for weight within a factor $O(n^{O(\log n)})$, gives $\widehat{G} \approx^{VC}_{\widehat{\epsilon}} \widehat{H}$ with high probability. Therefore $(\tilde{G} \setminus \widehat{G}) \cup \widehat{H} \approx^{VC}_{2\widehat{\epsilon}} G$ with high probability. Applying this inductively for $O(\log n)$ steps gives the desired relation by setting $\widehat{\epsilon} = \frac{\epsilon}{O(\log n)}$ as was done in the iterative routine above.

Sparsifying G requires $O(m \cdot \text{poly}(\log n, \epsilon^{-1}))$ work [55]. Furthermore, in Section 1.9.2 we will show that VERTEXSPARSIFY can be maintained dynamically in worst-case update time of $O(\text{poly}(\log n, \epsilon^{-1}))$, so it's static runtime must be $O(m \cdot \text{poly}(\log n, \epsilon^{-1}))$.

□

1.9.2 Dynamic Minimum Cut of Bipartite Graphs

Now that we have the full process of VERTEXSPARSIFY, we will give the dynamic algorithm for maintaining a $(1+\epsilon)$-approximate minimum cut in amortized $O(\text{poly}(\log n, \epsilon^{-1}))$ time. The algorithm in Figure 1.14 will be analogous to the one given in Section 1.7, but will replace sparsification of G_{VC} with VERTEXSPARSIFY, improving the approximation by a factor of 2.

1. Dynamically maintain a sparsified G, which we will denote $\tilde{G}$
2. Dynamically maintain a *branch vertex cover*, VC, on $\tilde{G}$, where we ensure $s, t \in VC$
3. Dynamically maintain a vertex sparsified $\tilde{G}$ using VC and $X\tilde{G} = V \setminus VC$ which we will denote H
4. Every $\frac{\epsilon}{2}\Delta_H(\widehat{S}_{V_H})$ dynamic steps, recompute $\widehat{S}_{V_H} \subset V_H$, an approximate minimum $s-t$ cut on H, ignoring all degree zero vertices

Figure 1.14: Dynamic $(1+\epsilon)$-approximate Minimum $s-t$ Cut

In this algorithm we run into the same issue of returning a cut of size $O(n)$ in amortized $O(\text{poly}(\log n, \epsilon^{-1}))$ time, and will allow a similar querying scheme. Let V_H be the non-zero degree vertex set of H. Our vertex sparsification process ensures that $VC \subseteq V_H$, so for the computed $\widehat{S}_{V_H} \subset V_H$, we will maintain the cut $\widehat{S}_{V_H} \cap VC \subset VC$ with $s \in \widehat{S}_{V_H}$. For a vertex $v \in VC$, return v is with s iff $v \in \widehat{S}_{V_H} \cap VC$, which takes $O(1)$ time. For a vertex $x \notin VC$, note that all of $N(x)$ must be in VC, and return that x is with s iff $w(x, \widehat{S}_{V_H} \cap VC) = w^{(x)}(\widehat{S}_{V_H} \cap VC)$ in $\tilde{G}$, taking $O(\text{poly}(\log n, \epsilon^{-1}))$ time to compute $w(x, \widehat{S}_{V_H} \cap VC)$ and $w^{(x)}(\widehat{S}_{V_H} \cap VC)$, by Corollary 1.6.2 and Corollary 1.7.4. Note that by restricting to VC we will be able take advantage of the approximation guarantees of vertex sparsification in the corollary below.

Corollary 1.9.9. *The dynamic algorithm maintains a $(1+\epsilon)$-approximate minimum $s-t$ cut in G, and will only compute an approximate minimum $s-t$ cut on H every $O(\epsilon OPT)$ dynamic steps, taking $O(OPT \cdot \text{poly}(\log n, \epsilon^{-1}))$ time each computation*

Proof. $\tilde{G} = F_1 \cup \ldots \cup F_K$ for some $K = O(\text{poly}(\log n, \epsilon^{-1}))$ by Corollary 1.6.2, so from Lemma 1.4.1 and Corollary 1.7.4, we know $|VC| = O(OPT \cdot \text{poly}(\log n, \epsilon^{-1}))$ and the degree of all vertices in $X\tilde{G}$ is $O(\text{poly}(\log n, \epsilon^{-1}))$. From Corollary 1.6.2, the weights of $\tilde{G}$ are in $[1, O(n)]$, and so property (1) of Theorem 1.9.1 implies that H has $O(OPT \cdot \text{poly}(\log n, \epsilon^{-1}))$ edges. Therefore, we can find a $(1+\widehat{\epsilon})$ approximate minimum

$s-t$ cut in H, in $O(OPT \cdot \text{poly}(\log n, \epsilon^{-1}))$ time.

Assume $\widehat{S}_{V_H} \subset V_H$ is returned as a $(1+\widehat{\epsilon})$-approximate minimum $s-t$ cut in H, with $\widehat{\epsilon} = \frac{\epsilon}{O(1)}$. Let $\widehat{S}_{VC} = \widehat{S}_{V_H} \cap VC$ be its restriction to VC, and let

$$\widehat{S} = \widehat{S}_{VC} \cup \{x \in X\tilde{G} : w(x, \widehat{S}_{VC}) = w^{(x)}(\widehat{S}_{VC})\}$$

be the extension of $\widehat{S}_{VC}$ onto $\tilde{G}$, which is the cut returned by our vertex querying scheme. From Corollary 1.6.2 and Theorem 1.9.1, we have $G \approx_{\widehat{\epsilon}} \tilde{G}$ and $\tilde{G} \approx_{\widehat{\epsilon}}^{VC} H$, respectively, which gives

$$\Delta_G(\widehat{S}) \leq (1+\widehat{\epsilon})\Delta_{\tilde{G}}(\widehat{S}) = (1+\widehat{\epsilon})\Delta_{\tilde{G}}(\widehat{S}_{VC}) \leq (1+\widehat{\epsilon})^2\Delta_H(\widehat{S}_{VC}).$$

On the other hand, let $\overline{S} \subset V$ be the minimum $s-t$ cut in G, and $\overline{S}_{VC} \subset VC$ be its restriction to VC. Using the fact that $\Delta_{\tilde{G}}(\overline{S}_{VC})$ is the weight of the minimal extension of $\overline{S}_{VC}$ in $\tilde{G}$, along with the approximations $G \approx_{\widehat{\epsilon}} \tilde{G}$ and $\tilde{G} \approx_{\widehat{\epsilon}}^{VC} H$ gives

$$\Delta_G(\overline{S}) \geq (1-\widehat{\epsilon})\Delta_{\tilde{G}}(\overline{S}) \geq (1-\widehat{\epsilon})\Delta_{\tilde{G}}(\overline{S}_{VC}) \geq (1-\widehat{\epsilon})^2\Delta_H(\overline{S}_{VC}).$$

The near-optimality of $\widehat{S}_{V_H}$ on H and setting $\widehat{S}_{VC} = \widehat{S}_{V_H} \cap VC$, gives,

$$\Delta_H(\overline{S}_{VC}) \geq (1-\widehat{\epsilon})\Delta_H(\widehat{S}_{V_H}) \geq (1-\widehat{\epsilon})\Delta_H(\widehat{S}_{VC})$$

Therefore, $\Delta_G(\widehat{S}) \leq (1+\widehat{\epsilon})^5\Delta_G(\overline{S})$, and by choosing $\widehat{\epsilon} = \frac{\epsilon}{O(1)}$ we maintain a $(1+\frac{\epsilon}{2})$-approximate minimum $s-t$ cut in G.

An approximate minimum $s-t$ cut on H will be re-computed in $\frac{\epsilon}{2}\Delta_H(\widehat{S}_{V_H})$ dynamic steps. $OPT = \Delta_G(\overline{S}) \leq (1+\epsilon)\Delta_H(\widehat{S}_{V_H})$, so $\Delta_H(\widehat{S}_{V_H}) = O(OPT)$

□

All that is left to be shown is that data structures can be maintained in $O(\text{poly}(\log n, \epsilon^{-1}))$ time per dynamic update. As a result of Corollary 1.6.2, it suffices to show the following

Theorem 1.9.10. *For each addition/deletion of an edge in $\tilde{G}$, maintaining $\widehat{G}$, H, and VC takes $O(\text{poly}(\log n, \epsilon^{-1}))$ time.*

As in Section 1.7.3, most of the necessary analysis for Theorem 1.9.10 will follow from the fact that all $x \in X\tilde{G}$ have degree $O(\text{poly}(\log n, \epsilon^{-1}))$, and the only substantial changes made to the data structures in one dynamic step, are done within the neighborhood of some $x \in X\tilde{G}$. We will also assume all of the dynamic data structure analysis of Section 1.7.3 with regards to maintaining a corresponding G_{VC} of some G.

In the rest of this section, we will first examine dynamically maintaining the pre-processing routine, particularly when vertices are moved in and out of the vertex cover. Then we will consider dynamically maintaining our vertex sparsification routine. Most of the time complexity analysis will follow from Section 1.7.3, and the only tricky part will be ensuring that dynamic changes do not multiply along iterations of the sparsification routine.

Maintaining $\widehat{G}$ As with the multi-graph G_{VC}, for $\widehat{G} \setminus X\tilde{G}$, an edge $e_\emptyset$ denotes an edge originally in $\tilde{G}$ and e_x denotes an edge that was moved into $\widehat{G} \setminus X\tilde{G}$ from N_x. For each $x \in X\tilde{G}$, let $x_{\max}$ denote the vertex such that $(x, x_{\max})$ has the maximum weight in N_x. Let $bucket(x)$ be the $i \in [L]$ such that $w(x, x_{\max}) \in [2^{i-1}, 2^i]$. We can use 1 as our scalar here because all weights of G are 1, so from Corollary 1.6.2, all weights of $\tilde{G}$ are in $[1, O(n)]$. In order to maintain each $x_{\max}$, we will assume that the data structure of $\tilde{G}$ is such that the adjacency list of each x is sorted by edge weight. Consequently, edge insertions/deletions in $\tilde{G}$ will require $O(\log n)$ time.

Maintaining each bucket for an edge insertion/deletion in $\tilde{G}$ will be analogous to maintaining G_{VC} in Section 1.7.3. We will first show that moving a vertex in and out of $X\tilde{G}$ can be done in $O(\text{poly}(\log n, \epsilon^{-1}))$ time, then give the overall update process, which will primarily just be composed of these two operations.

Lemma 1.9.11. *If v is not in VC, then running* REMOVEXG$(\widehat{G}, X\tilde{G}, v)$ *will output $\widehat{G}$ with $v \in VC$ in $O(deg_v \log n)$ time, where deg_v is the degree of v in $\tilde{G}$*

REMOVEXG($\tilde{G}, X\tilde{G}, v$)

1. Delete all edges e_v incident to $v_{\max}$ from $\widehat{G} \setminus X\tilde{G}$
2. Delete all edges incident to v from $\widehat{G}_{bucket(v)}$
3. For all edges e incident to v in $\tilde{G}$, add $e_\emptyset$ into $\widehat{G} \setminus X\tilde{G}$

Figure 1.15: Removing a Vertex from $X\tilde{G}$

Proof. Costs of the three steps are:

1. Deleting all edges e_v incident to $v_{\max}$ from $\widehat{G} \setminus X\tilde{G}$ takes $O(\log n)$ time per deletion and $O(deg_v)$ deletions.
2. Deleting all edges incident to v from $\widehat{G}_{bucket(v)}$ takes $O(\log n)$ time per deletion and $O(deg_v)$ deletions.
3. Adding $e_\emptyset$ into $\widehat{G} \setminus X\tilde{G}$ takes $O(\log n)$ time and is done for all edges e incident to v in $\tilde{G}$, so $O(deg_v)$ times

If v is not in VC, then v cannot be incident to any vertices in $X\tilde{G}$. Therefore, placing v in VC implies that v cannot be incident to any edges in all $\widehat{G}_k$ and no edges e_v exist in $\widehat{G} \setminus X\tilde{G}$. REMOVEXG($\widehat{G}, X\tilde{G}, v$) performs exactly these removals and inserts all necessary $e_\emptyset$ incident to v into $\widehat{G} \setminus X\tilde{G}$

□

Lemma 1.9.12. *If v is not in VC, but was placed in VC for $\widehat{G}$, then* INSERTXG($\widehat{G}, X\tilde{G}, v$) *will output $\widehat{G}$ with $v \notin VC$ in $O(deg_v \log n)$ time, where deg_v is the degree of v in $\tilde{G}$*

Proof. Costs of the two steps are:

1. Deleting all edges $e_\emptyset$ incident to v in $\widehat{G} \setminus X\tilde{G}$ takes $O(\log n)$ time per deletion and $O(deg_v)$ deletions.

INSERTXG$(\widehat{G}, X\tilde{G}, v)$

1. Delete all edges $e_\emptyset$ incident to v in $\widehat{G} \setminus X\tilde{G}$
2. For all edges $e = (v, w) \in \tilde{G}$ incident to v
   (a) If $w(v, w) < \frac{\epsilon}{d} w(v, v_{\max})$: insert $(w, v_{\max})_v$ into $\widehat{G} \setminus X\tilde{G}$
   (b) Otherwise: insert (v, w) into $\widehat{G}_{bucket(v)}$

Figure 1.16: Inserting a Vertex into $X\tilde{G}$

2. Checking if $w(v, w) < \frac{\epsilon}{d} w(v, v_{\max})$ and inserting $(w, v_{\max})_v$ into $\widehat{G} \setminus X$ or inserting (v, w) into $\widehat{G}_{bucket(v)}$ takes $O(\log n)$ time. This is done for all edges $e = (v, w) \in \tilde{G}$ incident to v, so $O(deg_v)$ times

If v is not in VC, but was placed in VC for $\widehat{G}$, then only edges $e_\emptyset$ are incident to v in $\widehat{G}$. Removing v from VC requires deleting all of these edges. Further, all edges e in N_v of sufficiently small weight must be moved to $\widehat{G} \setminus X\tilde{G}$ as e_v, and the rest of N_v must be placed in the appropriate $\widehat{G}_i$. INSERTXG$(\widehat{G}, X\tilde{G}, v)$ performs exactly these operations.

□

The full dynamic update process of $\widehat{G}$ for each $e = (u, v)$ insertion/deletion in $\tilde{G}$ will then be as follows.

1. For u and v, REMOVEXG$(\widehat{G}, X\tilde{G}, v)$ if $v \notin VC$
2. Update VC and $\tilde{G}$ as done in section 5
3. Add/delete $(u, v)_\emptyset$ from $\widehat{G} \setminus X\tilde{G}$
4. Update $u_{\max}$ and $v_{\max}$, which will simply require looking at the first edge incident to u and v in $\tilde{G}$, as the list is sorted by weight
5. For u and v, INSERTXG$(\widehat{G}, X\tilde{G}, v)$ if $v \notin VC$

Lemma 1.9.13. *For each edge addition/deletion in $\tilde{G}$, maintaining $\widehat{G} = \widehat{G} \setminus X\tilde{G} \cup \widehat{G}_1 \cup \ldots \cup \widehat{G}_L$ takes $O(\text{poly}(\log n, \epsilon^{-1}))$ time.*

Proof. Note that INSERTXG$(\widehat{G}, X\tilde{G}, v)$ and REMOVEXG$(\widehat{G}, X\tilde{G}, v)$ are only performed if $v \notin VC$, which implies that the degree of v in $\tilde{G}$ is $O(\text{poly}(\log n, \epsilon^{-1}))$. Updating VC and $\tilde{G}$ is known to take $O(\text{poly}(\log n, \epsilon^{-1}))$ time. Steps 3 and 4 clearly take $O(\log n)$ time. Therefore, the full runtime of this update process is $O(\text{poly}(\log n, \epsilon^{-1}))$.

□

Maintaining BoundedVertexSparsify We will dynamically sparsify the multi-graph $\widehat{G} \setminus X\tilde{G}$ as per usual, so each edge insertion/deletion requires $O(\text{poly}(\log n, \epsilon^{-1}))$ update time for $\widehat{G} \setminus X\tilde{G}$. Accordingly, we will only consider maintaining the necessary data structures for BOUNDEDVERTEXSPARSIFY of each $\widehat{G}_k$, which we will simply denote as G with bipartition (VC, XG).

Alterations to G are made by the dynamic update process in the previous section, which implies that we only need to consider the following changes to G. Add/Delete a vertex x from X, and add/delete N_x from G. Add/Delete an edge within N_x for some $x \in X$. If an edge is added/deleted from N_x, we will simply delete N_x from G, and then add N_x with the edge added/deleted to G. Accordingly, in order to establish that our data structures can be maintained in $O(\text{poly}(\log n, \epsilon^{-1}))$ update time, we just need to show that adding/deleting any N_x from G can be done in $O(\text{poly}(\log n, \epsilon^{-1}))$ update time.

For each level i of computing a light vertex set and running SAMPLE, we need to maintain G_i, G^i_{VC}, XG_i^{light} and all $F_{i,j}$ in F_i. The data structures for G_i and G^i_{VC} will be as in Subsection 1.7.3. Assume that the data structure of each $F_{i,j}$ is such that we can search for edges in $O(\log n)$-time, either by search trees or linked lists with back pointers (see e.g. [78], Chapters 10.2, 10.3, and 13). The data structure each XG_i^{light} will just be a list of vertices with insertion/deletion taking $O(\log n)$ time.

We will still assume edge additions/deletions in G_i, G^i_{VC} can be maintained in time

INSERTSTAR$(G_i, XG_i, XG_i^{light}, N_x)$

1. Update $G_i \leftarrow G_i \cup N_x$, $XG_i \leftarrow XG_i \cup x$, and insert K_x into G_{VC}^i
2. For the first $e_x \in K_x$ that can be added to some $F_{i,j}$: Update $F_{i,j} \leftarrow F_{i,j} \cup e_x$, $XG_i^{light} \leftarrow XG_i^{light} \cup x$, and remove K_x from G_{VC}^i
3. If no $e_x \in K_x$ can be added to any $F_{i,j}$, with probability $\frac{1}{2}$: run INSERTSTAR$(G_{i+1}, XG_{i+1}, XG_{i+1}^{light}, 2N_x)$

Figure 1.17: Add N_x to G_i

$O(\text{poly}(\log n, \epsilon^{-1}))$, as was shown in Subsection 1.7.3. Most of the time complexity analysis will then follow from this, and we just need to establish that the additions/deletions will not multiply as we move down the pipeline. This will ultimately follow from our construction of the t-clique forests.

Adding some N_x to G_i The algorithm in Figure 1.17 will add a vertex x to G_i, along with the corresponding N_x.

Lemma 1.9.14. INSERTSTAR$(G_i, XG_i, XG_i^{light}, N_x)$ *adds N_x to G_i while maintaining t-clique forest F_i*

Proof. If some $e_x \in K_x$ can be added to some $F_{i,j}$, then by construction, $F_i \cap K_x = e_x$ and $x \in XG_i^{light}$. Therefore, F_i is still a t-clique forest, and $x \in XG_i^{light}$ implies $x \notin XG_{i+1}$, so it is only necessary to add e_x to $F_{i,j}$ and x to XG_i^{light}.

If no $e_x \in K_x$ can be added to any $F_{i,j}$, then $F_i \cap K_x = \emptyset$ and $x \in XG_i^{heavy}$. Therefore, F_i is still a t-clique forest, and $x \in XG_i^{heavy}$ implies a coin must be flipped to determine whether x is added to XG_{i+1} and $2N_x$ is added to G_{i+1}.

□

Furthermore, we still maintain $G_{VC}^i = \bigcup_{x \in XG_i^{heavy}} K_x$

REMOVESTAR$(G_i, XG_i, XG_i^{light}, N_x)$

1. Update $G_i \leftarrow G_i \setminus N_x$, $XG_i \leftarrow XG_i \setminus x$, and remove K_x from G_{VC}^i
2. If some e_x is in some $F_{i,j}$
   (a) Update $F_{i,j} \leftarrow F_{i,j} \setminus e_x$, $XG_i^{light} \leftarrow XG_i^{light} \setminus x$
   (b) If some edge $f_y \in G_{VC}^i$ can be added to $F_{i,j}$
   - Update $F_{i,j} \leftarrow F_{i,j} \cup f_y$, $XG_i^{light} \leftarrow XG_i^{light} \cup y$, and remove K_y from G_{VC}^i
   - run REMOVESTAR$(G_{i+1}, XG_{i+1}, XG_{i+1}^{light}, 2N_y)$ if $y \in XG_{i+1}$
3. If no $e_x \in K_x$ is in any $F_{i,j}$, run REMOVESTAR$(G_{i+1}, XG_{i+1}, XG_{i+1}^{light}, 2N_x)$ if $x \in XG_{i+1}$

Figure 1.18: Remove N_x from G_i

Deleting some N_x from G_i The algorithm in Figure 1.18 will delete a vertex x from G_i, along with the corresponding N_x.

Lemma 1.9.15. REMOVESTAR$(G_i, XG_i, XG_i^{light}, N_x)$ *removes N_x from G_i while maintaining t-clique forest F_i*

Proof. If we had $F_i \cap K_x = e_x$, then x was in XG_i^{light}, so e_x must be removed from some $F_{i,j}$ and x must be removed from XG_i^{light}. F_i was a t-clique forest and $G_{VC}^i = \bigcup_{x \in XG_i^{heavy}} K_x$ (as was noted), implying that multiple edges in G_{VC}^i cannot be added to $F_{i,j}$ without creating a cycle. If f_y is added to $F_{i,j}$ then y is added to XG_i^{light} and $F_i \cap K_y = f_y$. Therefore, F_i is still a t-clique forest, and because $y \in XG_i^{light}$, it is now necessary to remove $2N_y$ from G_{i+1} if $y \in XG_{i+1}$.

If we have $F_i \cap K_x = \emptyset$, then $x \in XG_i^{heavy}$ and F_i is still a t-clique forest. Further $XG_{i+1} \subseteq XG_i$, so it is necessary to remove $2N_x$ from G_{i+1} if $x \in XG_{i+1}$.

□

Furthermore, we still maintain $G_{VC}^i = \bigcup_{x \in XG_i^{heavy}} K_x$

Lemma 1.9.16. *For any addition/deletion of some x from XG_0 and N_x from G_0, maintaining H takes $O(t \cdot \mathrm{poly}(\log n, \epsilon^{-1}))$ time*

Proof. Checking each forest for an edge insertion/deletion takes $O(t \log n)$ time. It follows almost immediately from the analysis in Subsection 1.7.3 that the rest of the computation in one iteration of INSERTSTAR and REMOVESTAR takes $O(\mathrm{poly}(\log n, \epsilon^{-1}))$ time. Furthermore, both can make at most one recursive call to themselves, so adding/deleting N_x from G_0 takes $O(l \cdot t \cdot \mathrm{poly}(\log n, \epsilon^{-1}))$ time where $l = O(\log n)$.

□

Proof of Theorem 1.9.10 : Any edge insertion/deletion in $\tilde{G}$ requires $O(\mathrm{poly}(\log n, \epsilon^{-1}))$ update time for $\widehat{G}$ and VC from Lemma 1.9.13. Therefore, there are at most $O(\mathrm{poly}(\log n, \epsilon^{-1}))$ additions/deletions of some N_x to some $\widehat{G}_i$, which will require $O(t \cdot \mathrm{poly}(\log n, \epsilon^{-1}))$ update time from Lemma 1.9.16, where $t = O(\mathrm{poly}(\log n, \epsilon^{-1}))$. Thus, the full dynamic update process of all data structures takes $O(\mathrm{poly}(\log n, \epsilon^{-1}))$ time per dynamic update of $\tilde{G}$.

1.10 Omitted Proofs of Section 1.5.2

In the following we give the omitted proofs of section Section 1.5.2, which mainly use standard arguments.

Lemma 1.5.5. *The output H of* LIGHT-SPECTRAL-SPARSIFY *is a $(1\pm\epsilon)$-spectral sparsifier with probability at least $1 - n^{-(c+1)}$ for any input graph G that is independent of the random choices of the algorithm.*

Proof. Let

$$R = \frac{\epsilon^2}{3(c+1)\ln n}.$$

For every edge $e \in G \setminus B$, let X_e be the random variable that is $4w_G(e) \cdot \mathcal{L}_e$ with probability

$1/4$ and 0 with probability $3/4$. We further set $\mathcal{L}_{B^{(j)}}$ for every $1 \leq j \leq \lceil 1/R \rceil$ as follows:

$$\mathcal{L}_{B_i^{(j)}} = \begin{cases} R \cdot \mathcal{L}_{B_i} & \text{if } 1 \leq j \leq \lfloor 1/R \rfloor \\ \mathcal{L}_{B_i} - \lfloor 1/R \rfloor R \cdot \mathcal{L}_{B_i} & \text{if } j = \lceil 1/R \rceil \end{cases}$$

Note that this definition simply guarantees that $\sum_{j=1}^{\lceil 1/R \rceil} \mathcal{L}_{B^{(j)}} = \mathcal{L}_{B_i}$ and $\mathcal{L}_{B_i^{(j)}} \leq R \cdot \mathcal{L}_{B_i}$ for every $1 \leq j \leq \lceil 1/R \rceil$. We now want to apply Theorem 1.5.4 with the random variables $Y = \sum_{e \in G \setminus B} X_e + \sum_{j=1}^{\lceil 1/R \rceil} \mathcal{L}_{B^{(j)}}$ and $Z = \mathcal{L}_G$. Observe that

$$\begin{aligned} \mathbb{E}[Y] &= E\left[\sum_{e \in G \setminus B} X_e + \sum_{j=1}^{\lceil 1/R \rceil} \mathcal{L}_{B^{(j)}}\right] \\ &= \sum_{e \in G \setminus B} \mathbb{E}[X_e] + \sum_{j=1}^{\lceil 1/R \rceil} \mathcal{L}_{B^{(j)}} \\ &= \sum_{e \in G \setminus B} \mathcal{L}_e + \mathcal{L}_B = \mathcal{L}_G = Z\,. \end{aligned}$$

For every edge $e \in G \setminus B$, using Lemma 1.5.3, we have

$$X_e \preceq 4w_G(e) \cdot \mathcal{L}_e \preceq \frac{\alpha}{t} \cdot \mathcal{L}_G \leq R \cdot \mathcal{L}_G\,.$$

Furthermore, using $B \preceq G$, we have

$$\mathcal{L}_{B_i^{(j)}} \leq R \cdot \mathcal{L}_{B_i} \preceq R \cdot \mathcal{L}_{G_{i-1}}$$

for every $1 \leq j \leq \lceil 1/R \rceil$. Thus, the preconditions of Theorem 1.5.4 are satisfied. We conclude that we have $\mathcal{L}_{G_H} \preceq (1+\epsilon)\mathcal{L}_G$ with probability at least

$$n \cdot \exp(-\epsilon^2/2R) \geq n \cdot \exp((c+1)\ln n) = 1/n^{c+1}\,.$$

A symmetric argument can be used for $(1-\epsilon)\mathcal{L}_G \preceq \mathcal{L}_H$. □

Lemma 1.5.6. *The output H of algorithm* SPECTRAL-SPARSIFY *is a $(1 \pm \epsilon)$-spectral sparsifier with probability at least $1 - 1/n^{c+1}$ for any input graph G that is independent of the random choices of the algorithm.*

Proof. Note that since $H = \bigcup_{i=1}^{k} B_i \cup G_k$ we have

$$\mathcal{L}_H = \mathcal{L}_{G_k} + \sum_{i=1}^{k} \mathcal{L}_{B_i} .$$

We now prove by induction on j that $\mathcal{L}_{G_k} + \sum_{i=k-j+1}^{k} \mathcal{L}_{B_i} \preceq (1 + \epsilon/(2k))^j \mathcal{L}_{G_{k-j}}$. This claim is trivially true for $j = 0$. For $1 \leq j \leq k$, we use the induction hypothesis and Lemma 1.5.5, which both hold with high probability, to get

$$\begin{aligned}
\mathcal{L}_{G_k} + \sum_{i=k-j+1}^{k} \mathcal{L}_{B_i} &= \mathcal{L}_{G_k} + \sum_{i=k-j+2}^{k} \mathcal{L}_{B_i} + \mathcal{L}_{B_{k-j+1}} \\
&\preceq (1 + \epsilon/(2k))^{j-1} \mathcal{L}_{G_{k-j+1}} + \mathcal{L}_{B_{k-j+1}} \\
&\preceq (1 + \epsilon/(2k))^{j-1} (\mathcal{L}_{G_{k-j+1}} + \mathcal{L}_{B_{k-j+1}}) \\
&\preceq (1 + \epsilon/(2k))^{j} \mathcal{L}_{G_{k-j}} .
\end{aligned}$$

We now have $\mathcal{L}_H \preceq (1 + \epsilon/(2k))^k \mathcal{L}_G$ with high probability by setting $j = k$. Using symmetric arguments we can prove $(1 - \epsilon/(2k))^k \mathcal{L}_G \preceq \mathcal{L}_H$. Since $(1 - \epsilon/(2k))^k \geq 1 - \epsilon$ and $(1 + \epsilon/(2k))^k \leq 1 + \epsilon$, the claim follows. □

Lemma 1.5.7. *With probability at least $1 - 2n^{-c}$, the number of iterations before algorithm* SPECTRAL-SPARSIFY *terminates is*

$$\min\{\lceil \log \rho \rceil, \lceil \log m/((c+1)\log n) \rceil\}.$$

Moreover the size of H is

$$O\left(\sum_{1\leq j\leq i}|B_i| + m/\rho + c\log n\right),$$

and the size of the third output of the graph is at most $\max\{O(c\log n), O(m/\rho)\}$.

Proof. We will show that, with probability $1-2n^{-c+1}$, every iteration j computes a graph G_{j+1} with half the number of edges in G_j. By a union bound, the probability that this fails to be true for any $j < n$ is at most $2n^{-c}$. This implies all claims.

We use the following standard Chernoff bound: Let $X = \sum_{k=1}^{N} X_k$, where $X_k = 1$ with probability p_k and $X_k = 0$ with probability $1-p_k$, and all X_k are independent. Let $\mu = \mathbb{E}[X] = \sum_{k=1}^{N} p_k$. Then $\mathbb{P}[X \geq (1+\delta)\mu] \leq \exp(-\frac{\delta^2}{2+\delta}\mu)$ for all $\delta > 0$.

We apply this bound on the output of LIGHT-SPECTRAL-SPARSIFY for every j. Concretely, we assign a random variable to each edge e of G_j, with $X_e = 1$ if and only if e is added to G_{j+1}. Then $\mathbb{E}[X] = N/4$. By construction, the number of edges in G_j is $N \geq (c+1)\log n$. Applying the Chernoff bound with $\delta = 2$ we get

$$\mathbb{P}[X \geq 2N] \leq \frac{1}{e^{N/4}} \leq \frac{1}{e^{((c+1)\log n)/4}} = \frac{1}{e^{1/4}n^{c+1}} \leq \frac{1}{2n^{c+1}}.$$

□

1.11 Guarantees of Combinatorial Reductions

We show some of the structural results necessary for the reductions in Sections 1.7, 1.8, and 1.9. We first show the guarantees of K_x:

Proof. (of Theorem 1.7.9) For any $x \in X$ and $S_{VC} \subset VC$, let $w_{K_x}(S_{VC})$ denote the weight of cutting S_{VC} in K_x. Consequently, for any $S_{VC} \subset VC$, $\Delta_{G_{VC}}(S_{VC}) = \Delta_{G\setminus X}(S_{VC}) + \sum_{x\in X} w_{K_x}(S_{VC})$, and it suffices to show that for all $x \in X$, $\frac{1}{2}w^{(x)}(S_{VC}) \leq w_{K_x}(S_{VC}) \leq w^{(x)}(S_{VC})$. □

Lemma 1.11.1. *For any $x \in X$ and $S \subset VC$, we have $\frac{1}{2} w_{K_x}(S) \leq w^{(x)}(S) \leq w_{K_x}(S)$*

Proof. Without loss of generality, assume $w(x,S) \leq w(x, VC \backslash S)$, so $w^{(x)}(S) = w(x,S) = \sum_{u \in S \cap N(x)} w(x,u)$

$$w_{K_x}(S) = \sum_{u \in S \cap N(x)} \sum_{v \in (VC \backslash S) \cap N(x)} \frac{w(x,u)w(x,v)}{\sum_{i \in N(x)} w(x,i)} = \sum_{u \in S \cap N(x)} w(x,u) \frac{w(x, VC \setminus S)}{\sum_{i \in N(x)} w(x,i)}$$

where by definition $\sum_{i \in N(x)} w(x,i) = w(x,S) + w(x, VC \setminus S)$ and so by assumption

$$\frac{1}{2} \leq \frac{w(x, VC \setminus S)}{\sum_{i \in N(x)} w(x,i)} \leq 1$$

□

Proof. (of Lemma 1.7.10) Each edge in $(u,v)_x \in G_{VC}$ has weight

$$w_{(u,v)_x} = \frac{w(x,v)w(x,u)}{\sum_{i \in N(x)} w(x,i)}$$

, $w(x,v)w(x,u) \geq \gamma^2$ and $\sum_{i \in N(x)} w_{(x,i)} \leq \gamma U d$. Also, we further note that $\sum_{i \in N(x)} w(x,i) \geq \max\{w(x,v)w(x,u)\}$, implying

$$\frac{w(x,v)w(x,u)}{\sum_{i \in N(x)} w(x,i)} \leq \frac{\max\{w(x,v)w(x,u)\}^2}{\sum_{i \in N(x)} w(x,i)} \leq \max\{w(x,v)w(x,u)\}$$

□

Next we bound the size of the vertex cover formed by removing all leaves, compared to the optimum.

Proof. (of Lemma 1.7.2) From [77, 79], given a tree T_0 with root r_0, leaves $l(T_0)$, and parents of the leaves $p(T_0)$, the greedy algorithm of taking $p(T_0)$ and iterating on $T_1 = T_0 \setminus \{l(T_0) \cup p(T_0)\}$, with $r_1 = r_0$ or r_1 arbitrary if $r_0 \in p(T_0)$, will give a minimum

vertex cover of T_0. If T_1 is a forest, iterate on each tree of the forest, where r_0 is the root of whichever tree it is contained in, and the remaining trees are arbitrarily rooted. Assume that if $T_i = r_i$ for some i, then $p(T_i) = \emptyset$.

Set $T = T_0$ and $r = r_0$, and suppose T_0 can be decomposed into $T_0 \dots T_d$ as above. Therefore, $\bigcup_{i=0}^{d} p(T_i)$ is a minimum vertex cover, and VC is $p(T_d) \cup \bigcup_{i=0}^{d-1}(p(T_i) \cup l(T_{i+1}))$

By construction, all $p(T_i)$ and $l(T_j)$ are disjoint, and we claim that $|p(T_i)| \geq |l(T_{i+1})|$ for all i. Assume T_i is a tree, and this will clearly still hold if T_i is a collection of disjoint trees. Each vertex in $l(T_{i+1})$ was not a leaf in T_i and is now a leaf in T_{i+1}. Further, $r_i \notin l(T_{i+1})$ because if $r_i \in T_{i+1}$, then $r_{i+1} = r_i$. Therefore, each vertex in $l(T_{i+1})$ must have had its degree reduced by removing $l(T_i)$ and $p(T_i)$. A vertex in $l(T_{i+1})$ cannot be connected to a vertex in $l(T_i)$ because then it would be in $p(T_i)$. Consequently, it must be connected to some vertex in $p(T_i)$, and if $|p(T_i)| < |l(T_{i+1})|$, then two vertices in $l(T_{i+1})$ must be connected to the same vertex in $p(T_i)$, creating a cycle in T_i, giving a contradiction. Thus

$$|VC| = p(T_d) + \sum_{i=0}^{d-1}(|p(T_i)| + |l(T_{i+1})|) \leq p(T_d) + \sum_{i=0}^{d-1} 2|p(T_i)| \leq 2\sum_{i=0}^{d} |p(T_i)| = 2|MVC|$$

□

CHAPTER 2

DETERMINANT-PRESERVING SPARSIFICATION OF SDDM MATRICES

This was joint work with John Peebles, Richard Peng, and Anup B. Rao.

2.1 Abstract

We show variants of spectral sparsification routines can preserve the total spanning tree counts of graphs, which by Kirchhoff's matrix-tree theorem, is equivalent to determinant of a graph Laplacian minor, or equivalently, of any SDDM matrix. Our analyses utilize this combinatorial connection to bridge between statistical leverage scores / effective resistances and the analysis of random graphs by [Janson, Combinatorics, Probability and Computing '94].

This leads to a routine that in quadratic time, sparsifies a graph down to about $n^{1.5}$ edges in ways that preserve both the determinant and the distribution of spanning trees (provided the sparsified graph is viewed as a random object).

Extending this algorithm to work with Schur complements and approximate Cholesky factorizations leads to algorithms for counting and sampling spanning trees which are nearly optimal for dense graphs. Specifically, we give an algorithm that computes a $(1 \pm \delta)$ approximation to the determinant of any SDDM matrix with constant probability in about $n^2\delta^{-2}$ time. This is the first routine for graphs that outperforms general-purpose routines for computing determinants of arbitrary matrices. We also give an algorithm that generates in about $n^2\delta^{-2}$ time a spanning tree of a weighted undirected graph from a distribution with total variation distance of δ from the w-uniform distribution .

2.2 Introduction

The determinant of a matrix is a fundamental quantity in numerical algorithms due to its connection to the rank of the matrix and its interpretation as the volume of the ellipsoid corresponding of the matrix. For graph Laplacians, which are at the core of spectral graph theory and spectral algorithms, the matrix-tree theorem gives that the determinant of the minor obtained by removing one row and the corresponding column equals to the total weight of all the spanning trees in the graph [80] . Formally on a weighted graph G with n vertices we have:

$$\det\left(\boldsymbol{L}^{G}_{1:n-1,1:n-1}\right) = \mathcal{T}_G,$$

where $\boldsymbol{L}^G$ is the graph Laplacian of G and and $\mathcal{T}_G$ is the total weight of all the spanning trees of G. As the all-ones vector is in the null space of $\boldsymbol{L}^G$, we need to drop its last row and column and work with $\boldsymbol{L}^{G}_{1:n-1,1:n-1}$, which is precisely the definition of SDDM matrices in numerical analysis [40]. The study of random spanning trees builds directly upon this connection between tree counts and determinants, and also plays an important role in graph theory [81, 82, 64].

While there has been much progress in the development of faster spectral algorithms, the estimation of determinants encapsulates many shortcomings of existing techniques. Many of the nearly linear time algorithms rely on sparsification procedures that remove edges from a graph while provably preserving the Laplacian matrix as an operator, and in turn, crucial algorithmic quantities such as cut sizes, Rayleigh quotients, and eigenvalues. The determinant of a matrix on the other hand is the product of all of its eigenvalues. As a result, a worst case guarantee of $1 \pm (\epsilon/n)$ per eigenvalue is needed to obtain a good overall approximation, and this in turn leads to additional factors of n in the number of edges needed in the sparse approximate.

Due to this amplification of error by a factor of n, previous works on numerically approximating determinants without dense-matrix multiplications [83, 84, 85] usually focus

on the log-determinant, and (under a nearly-linear running time) give errors of additive ϵn in the log determinant estimate, or a multiplicative error of $\exp(\epsilon n)$ for the determinant. The lack of a sparsification procedure also led to the running time of random spanning tree sampling algorithms to be limited by the sizes of the dense graphs generated in intermediate steps [86, 87, 88].

In this paper, we show that a slight variant of spectral sparsification preserves determinant approximations to a much higher accuracy than applying the guarantees to individual edges. Specifically, we show that sampling $\omega(n^{1.5})$ edges from a distribution given by leverage scores, or weight times effective resistances, produces a sparser graph whose determinant approximates that of the original graph. Furthermore, by treating the sparsifier itself as a random object, we can show that the spanning tree distribution produced by sampling a random tree from a random sparsifier is close to the spanning tree distribution in the original graph in total variation distance. Combining extensions of these algorithms with sparsification based algorithms for graph Laplacians then leads to quadratic time algorithms for counting and sampling random spanning trees, which are nearly optimal for dense graphs with $m = \Theta(n^2)$.

This determinant-preserving sparsification phenomenon is surprising in several aspects: because we can also show—both experimentally and mathematically—that on the complete graph, about $n^{1.5}$ edges are necessary to preserve the determinant, this is one of the first graph sparsification phenomenons that requires the number of edges to be between $>> n$. The proof of correctness of this procedure also hinges upon combinatorial arguments based on the matrix-tree theorem in ways motivated by a result for Janson for complete graphs [4], instead of the more common matrix-concentration bound based proofs [55, 57, 89, 90]. Furthermore, this algorithm appears far more delicate than spectral sparsification: it requires global control on the number of samples, high quality estimates of resistances (which is the running time bottleneck in Theorem 2.5.1 below), and only holds with constant probability. Nonetheless, the use of this procedure into our determinant estimation and spanning tree

generation algorithms still demonstrates that it can serve as a useful algorithmic tool.

2.2.1 Our Results

We will use $G = (V, E, w)$ to denote weighted multigraphs, and $\boldsymbol{d}_u \stackrel{\text{def}}{=} \sum_{e:e\ni u} w_e$ to denote the weighted degree of vertex u. The weight of a spanning tree in a weighed undirected multigraph is:

$$w(T) \stackrel{\text{def}}{=} \prod_{e\in T} w_e.$$

We will use $\mathcal{T}_G$ to denote the total weight of trees, $\mathcal{T}_G \stackrel{\text{def}}{=} \sum_{T\in\mathcal{T}} w(T)$. Our key sparsification result can be described by the following theorem:

Theorem 2.2.1. *Given any graph G and any parameter δ, we can compute in $O(n^2\delta^{-2})$ time a graph H with $O(n^{1.5}\delta^{-2})$ edges such that with constant probability we have*

$$(1-\delta)\,\mathcal{T}_G \leq \mathcal{T}_H \leq (1+\delta)\,\mathcal{T}_G.$$

This implies that graphs can be sparsified in a manner that preserves the determinant, albeit to a density that is not nearly-linear in n.

We show how to make our sparsification routine to errors in estimating leverage scores, and how our scheme can be adapted to implicitly sparsify dense objects that we do not have explicit access to. In particular, we utilize tools such as rejection sampling and high quality effective resistance estimation via projections to extend this routine to give determinant-preserving sparsification algorithms for Schur complements, which are intermediate states of Gaussian elimination on graphs, using ideas from the sparsification of random walk polynomials.

We use these extensions of our routine to obtain a variety of algorithms built around our graph sparsifiers. Our two main algorithmic applications are as follows. We achieve the first algorithm for estimating the determinant of an SDDM matrix that is faster than general

purpose algorithms for the matrix determinant problem. Since the determinant of an SDDM m corresponds to the determinant of a graph Laplacian with one row/column removed.

Theorem 2.2.2. *Given an SDDM matrix $\boldsymbol{M}$, there is a routine* DETAPPROX *which in $\widetilde{O}\left(n^2\delta^{-2}\right)$ time outputs D such that $D = (1 \pm \delta)\det(\boldsymbol{M})$ with high probability*

A crucial thing to note which distinguishes the above guarantee from most other similar results is that we give a multiplicative approximation of the $\det(M)$. This is much stronger than giving a multiplicative approximation of $\log\det(M)$, which is what other work typically tries to achieve.

The sparsifiers we construct will also approximately preserve the spanning tree distribution, which we leverage to yield a faster algorithm for sampling random spanning trees. Our new algorithm improves upon the current fastest algorithm for general weighted graphs when one wishes to achieve constant—or slightly sub-constant—total variation distance.

Theorem 2.2.3. *Given an undirected, weighted graph $G = (V, E, w)$, there is a routine* APPROXTREE *which in expected time $\widetilde{O}\left(n^2\delta^{-2}\right)$ outputs a random spanning tree from a distribution that has total variation distance $\leq \delta$ from the w-uniform distribution on G.*

2.2.2 Prior Work

Graph Sparsification

In the most general sense, a graph sparsification procedure is a method for taking a potentially dense graph and returning a sparse graph called a *sparsifier* that approximately still has many of the same properties of the original graph. It was introduced in [91] for preserving properties related to minimum spanning trees, edge connectivity, and related problems. [92] defined the notion of *cut sparsification* in which one produces a graph whose cut sizes approximate those in the original graph. [93] defined the more general notion of *spectral sparsification* which requires that the two graphs' Laplacian matrices approximate each

other as quadratic forms.[1] In particular, this spectral sparsification samples $\widetilde{O}(n/\epsilon^2)$ edges from the original graph, yielding a graph with $\widetilde{O}(n/\epsilon^2)$ whose quadratic forms—and hence, eigenvalues—approximate each other within a factor of $(1 \pm \epsilon)$. This implies that their determinants approximate each other within $(1 \pm \epsilon)^n$. This is not useful from the perspective of preserving the determinant: since one would need to samples $\Omega(n^3)$ edges to get a constant factor approximation, one could instead exactly compute the determinant or sample spanning trees using exact algorithms with this runtime.

All of the above results on sparsification are for undirected graphs. More recently, [94] has defined a useful notion of sparsification for directed graphs along with a nearly linear time algorithm for constructing sparsifiers under this notion of sparsification.

Determinant Estimation

Exactly calculating the the determinant of an arbitrary matrix is known to be equivalent to matrix multiplication [95]. For approximately computing the log of the determinant, [96] uses the identity $\log(\det(A)) = \text{tr}(\log(B)) + \text{tr}(\log(B^{-1}A))$ to do this whenever one can find a matrix B such that the $\text{tr}(\log(B)) = \log(\det(B))$ and $\text{tr}(\log(B^{-1}A)) = \log(\det(B^{-1}A)$ can both be quickly approximated.[2]

For the special case of approximating the log determinant of an SDD matrix, [97] applies this same identity recursively where the B matrices are a sequence of ultrasparsifiers inspired by the recursive preconditioning framework of [40]. They obtain a running time of $O(m(n^{-1}\epsilon^{-2} + \epsilon^{-1})\text{polylog}(n\kappa/\epsilon))$ for estimating the log determinant to additive error ϵ.

[98] estimates the log determinant of arbitrary positive definite matrices, but has runtime that depends linearly on the condition number of the matrix.

In contrast, our work is the first we know of that gives a multiplicative approximation of the determinant itself, rather than its log. Despite achieving a much stronger approximation

[1] If two graphs Laplacian matrices approximate each other as quadratic forms then their cut sizes also approximate each other.

[2] Specifically, they take B as the diagonal of A and prove sufficient conditions for when the log determinant of $B^{-1}A$ can be quickly approximated with this choice of B.

guarantee, our algorithm has essentially the same runtime as that of [97] when the graph is dense. Note also that if one wishes to conduct an "apples to apples" comparison by setting their value of ϵ small enough in order to match our approximation guarantee, their algorithm would only achieve a runtime bound of $O(mn\delta^{-2}\text{polylog}(n\kappa/\epsilon))$, which is never better than our runtime and can be as bad as a factor of n worse.[3]

Sampling Spanning Trees

Previous works on sampling random spanning trees are a combination of two ideas: that they could be generated using random walks, and that they could be mapped from a random integer via Kirchoff's matrix tree theorem. The former leads to running times of the form $O(nm)$ [99, 100], while the latter approach[101, 102, 103, 104] led to routines that run in $O(n^\omega)$ time, where $\omega \approx 2.373$ is the matrix multiplication exponent [105].

These approaches have been combined in algorithms by Kelner and Madry [86] and Madry, Straszak and Tarnawski [87]. These algorithms are based on simulating the walk more efficiently on parts of the graphs, and combining this with graph decompositions to handle the more expensive portions of the walks globally. Due to the connection with random-walk based spanning tree sampling algorithms, these routines often have inherent dependencies on the edge weights. Furthermore, on dense graphs their running times are still worse than the matrix-multiplication time routines.

However, recent work after the publication of this result improved upon these techniques to simulate random walks with clever ball growing techniques and amortization of the costs to achieve an almost linear random sampling procedure for weighted graphs [106]. Additionally, this random sampling does not incur any error in the distribution. While this result does supersede our result on spanning tree generation, it uses substantially different techniques and does not apply to determinant sparsification or computation.

[3]This simplification of their runtime is using the substitution $\epsilon = \delta/n$ which gives roughly $(1 \pm \delta)$ multiplicative error in estimating the determinant for their algorithm. This simplification is also assuming $\delta \leq 1$, which is the only regime we analyze our algorithm in and thus the only regime in which we can compare the two.

When this work was published, the previous best running time for generating a random spanning tree from a weighted graph was $\widetilde{O}\left(n^{5/3}m^{1/3}\log^2\left(1/\delta\right)\right)$ in [88]. It works by combining a recursive procedure similar to those used in the more recent $O(n^\omega)$ time algorithms [104] with spectral sparsification ideas, achieving a runtime of $\widetilde{O}(n^{5/3}m^{13})$. When $m = \Theta\left(n^2\right)$, the algorithm in [88] takes $\widetilde{O}\left(n^{7/3}\right)$ time to produce a tree from a distribution that is $o(1)$ away from the w-uniform distribution, which is slower by nearly a $n^{1/3}$ factor than the algorithm given in this paper.

Our algorithm can be viewed as a natural extension of the sparsification0-based approach from [88]: instead of preserving the probability of a single edge being chosen in a random spanning tree, we instead aim to preserve the entire distribution over spanning trees, with the sparsifier itself also considered as a random variable. This allow us to significantly reduce the sizes of intermediate graphs, but at the cost of a higher total variation distance in the spanning tree distributions. This characterization of a random spanning tree is not present in any of the previous works, and we believe it is an interesting direction to combine our sparsification procedure with the other algorithms.

2.2.3 Organization

Section 2.3 will introduce the necessary notation and some of the previously known fundamental results regarding the mathematical objects that we work with throughout the paper. Section 2.4 will give a high-level sketch of our primary results and concentration bounds for total tree weight under specific sampling schemes. Section 2.5 leverages these concentration bounds to give a quadratic time sparsification procedure (down to $\Omega(n^{1.5})$ edges) for general graphs. Section 2.6 uses random walk connections to extend our sparsification procedure to the Schur complement of a graph. Section 2.7 utilizes the previous routines to achieve a quadratic time algorithm for computing the determinant of SDDM matrices. Section 2.8 combines our results and modifies previously known routines to give a quadratic time algorithm for sampling random spanning trees with low total variation distance. Section 2.9

extends our concentration bounds to random samplings where an arbitrary tree is fixed, and is necessary for the error accounting of our random spanning tree sampling algorithm. Section 2.10 proves the total variation distance bounds given for our random sampling tree algorithm.

2.3 Background

2.3.1 Graphs, Matrices, and Random Spanning Trees

The goal of generating a random spanning tree is to pick tree T with probability proportional to its weight, which we formalize in the following definition.

Definition 2.3.1 (w-uniform distribution on trees)**.** Let $\boldsymbol{Pr}_T^G(\cdot)$ be a probability distribution on $\mathcal{T}_G$ such that

$$\boldsymbol{Pr}_T^G\left(T = T_0\right) = \frac{\Pi_{e \in T_0} w_e}{\mathcal{T}_G}.$$

We refer to $\boldsymbol{Pr}_T^G(\cdot)$ as the w-uniform distribution on the trees of G.

When the graph G is unweighted, this corresponds to the uniform distribution on $\mathcal{T}_G$.

We refer to $\boldsymbol{Pr}_T^G(\cdot)$ as the w-uniform distribution on $\mathcal{T}_G$. When the graph G is unweighted, this corresponds to the uniform distribution on $\mathcal{T}_G$. Furthermore, as we will manipulate the probability of a particular tree being chosen extensively, we will denote such probabilities with $\boldsymbol{Pr}^G(\widehat{T})$, aka:

$$\boldsymbol{Pr}^G\left(\widehat{T}\right) \stackrel{\text{def}}{=} \boldsymbol{Pr}_T^G\left(T = \widehat{T}\right).$$

The Laplacian of a graph $G = (V, E, w)$ is an $n \times n$ matrix specified by:

$$\boldsymbol{L}_{uv} \stackrel{\text{def}}{=} \begin{cases} \boldsymbol{d}_u & \text{if } u = v \\ -w_{uv} & \text{if } u \neq v \end{cases}$$

We will write $\boldsymbol{L}^G$ when we wish to indicate which graph G that the Laplacian corresponds to and $\boldsymbol{L}$ when the context is clear. When the graph has multi-edges, we define w_{uv} as the sum of weights of all the edges e that go between vertices u, v. Laplacians are natural objects to consider when dealing with random spanning trees due to the matrix tree theorem, which states that the determinant of $\boldsymbol{L}$ with any row/column corresponding to some vertex removed is the total weight of spanning trees. We denote this removal of a vertex u as $\boldsymbol{L}_{-u}$. As the index of vertex removed does not affect the result, we will usually work with $\boldsymbol{L}_{-n}$. Furthermore, we will use $\det(\boldsymbol{M})$ to denote the determinant of a matrix. As we will work mostly with graph Laplacians, it is also useful for us to define the 'positive determinant' $\det_+$, where we remove the last row and column. Using this notation, the matrix tree theorem can be stated as:

$$\mathcal{T}_G = \det(\boldsymbol{L}^G_{-n}) = \det_+\left(\boldsymbol{L}^G\right).$$

We measure the distance between two probability distributions by total variation distance.

Definition 2.3.2. Given two probability distributions p and q on the same index set Ω, the *total variation distance* between p and q is given by

$$d_{TV}(p,q) \stackrel{\text{def}}{=} \frac{1}{2}\sum_{x\in\Omega}|p(x)-q(x)|.$$

Let $G = (V, E, w)$ be a graph and $e \in E$ an edge. We write G/e to denote the graph obtained by contracting the edge e, i.e., identifying the two endpoints of e and deleting any self loops formed in the resulting graph. We write $G\backslash e$ to denote the graph obtained by deleting the edge e from G. We extend these definitions to G/F and $G\backslash F$ for $F \subseteq E$ to refer to the graph obtained by contracting all the edges in F and deleting all the edges in F, respectively.

Also, for a subset of vertices V_1, we use $G[V_1]$ to denote the graph induced on the vertex of V_1. letting $G(V_1)$ be the edges associated with $\boldsymbol{L}_{[V_1,V_1]}$ in the Schur complement.

2.3.2 Effective Resistances and Leverage Scores

The matrix tree theorem also gives connections to another important algebraic quantity: the *effective resistance* between two vertices. This quantity is formally given as $\mathcal{R}_{eff}(u,v) \stackrel{\text{def}}{=} \boldsymbol{\chi}_{uv}^{\intercal} \boldsymbol{L}^{-1} \boldsymbol{\chi}_{uv}$ where $\boldsymbol{\chi}_{uv}$ is the indicator vector with 1 at u, -1 at v, and 0 everywhere else. Via the adjugate matrix, it can be shown that the effective resistance of an edge is precisely the ratio of the number of spanning trees in G/e over the number in G:

$$\mathcal{R}_{eff}(u,v) = \frac{\mathcal{T}_{G/e}}{\mathcal{T}_G}.$$

As $w_e \cdot \mathcal{T}_{G/e}$ is the total weight of all trees in G that contain edge e, the fraction[4] of spanning trees that contain $e = uv$ is given by $w_e \mathcal{R}_{eff}(u,v)$. This quantity is called the *statistical leverage score* of an edge, and we denote it by $\overline{\boldsymbol{\tau}}_e$. It is fundamental component of many randomized algorithms for sampling / sparsifying graphs and matrices [55, 107, 57].

The fact that $\overline{\boldsymbol{\tau}}_e$ is the fraction of trees containing e also gives one way of deriving the sum of these quantities:

Fact 2.3.3. *(Foster's Theorem) On any graph G we have*

$$\sum_e \overline{\boldsymbol{\tau}}_e = n - 1.$$

The resistance $\mathcal{R}_{eff}(u,v)$, and in turn the statistical leverage scores $\overline{\boldsymbol{\tau}}_e$ can be estimated using linear system solves and random projections [55]. For simplicity, we follow the abstraction utilized by Madry, Straszak, and Tarnawski [87], except we also allow the intermediate linear system solves to utilize a sparsifier instead of the original graph.

Lemma 2.3.4. *(Theorem 2.1. of [87])*

Let $G = (V, E)$ be a graph with m edges. For every $\epsilon > 0$ we can find in $\tilde{O}(\min\{m\epsilon^{-2}, m + n\epsilon^{-4}\})$ time an embedding of the effective resistance metric into $\Re^{O(\epsilon^{-2}\log m)}$ such that with

[4] provided one thinks of an edge with weight w as representing w parallel edges, or equivalently, counts spanning trees with multiplicity according to their weight

high probability allows one to compute an estimate $\widetilde{\mathcal{R}}_{eff}(u,v)$ of any effective resistance satisfying

$$\forall u, v \in V \qquad (1-\epsilon)\,\widetilde{\mathcal{R}}_{eff}\,(u,v) \leq \mathcal{R}_{eff}\,(u,v) \leq (1+\epsilon)\,\widetilde{\mathcal{R}}_{eff}\,(u,v)\,.$$

Specifically, each vertex u in this embedding is associated with an (explicitly stored) $\boldsymbol{z}_u \in \Re^{O(\epsilon^{-2}\log m)}$, and for any pair of vertices, the estimate $\widetilde{\mathcal{R}}_{eff}(u,v)$ is given by:

$$\widetilde{\mathcal{R}}_{eff}\,(u,v) = \|\boldsymbol{z}_u - \boldsymbol{z}_v\|_2^2\,,$$

which takes $O(\epsilon^{-2}\log m)$ time to compute once we have the embedding.

2.3.3 Schur Complements

For our applications, we will utilize our determinant-preserving sparsification algorithms in recursions based on Schur complements. A partition of the vertices, which we will denote using

$$V = V_1 \sqcup V_2,$$

partitions the corresponding graph Laplacian into blocks which we will denote using indices in the subscripts:

$$\boldsymbol{L} = \begin{bmatrix} \boldsymbol{L}_{[V_1,V_1]} & \boldsymbol{L}_{[V_1,V_2]} \\ \boldsymbol{L}_{[V_2,V_1]} & \boldsymbol{L}_{[V_2,V_2]} \end{bmatrix}.$$

The Schur complement of G, or $\boldsymbol{L}$, onto V_1 is then:

$$\textsc{Sc}\,(G,V_1) = \textsc{Sc}\left(\boldsymbol{L}^G, V_1\right) \stackrel{\text{def}}{=} \boldsymbol{L}^G_{[V_1,V_1]} - \boldsymbol{L}^G_{[V_1,V_2]}\left(\boldsymbol{L}^G_{[V_2,V_2]}\right)^{-1}\boldsymbol{L}^G_{[V_2,V_1]},$$

and we will use $\textsc{Sc}\,(G,V_1)$ and $\textsc{Sc}\left(\boldsymbol{L}^G,V_1\right)$ interchangeably. We further note that we will always consider V_1 to be the vertex set we Schur complement onto, and V_2 to be the vertex set we eliminate, except for instances in which we need to consider both $\textsc{Sc}\,(G,V_1)$ and

$\mathrm{Sc}(G, V_2)$.

Schur complements behave nicely with respect to determinants determinants, which suggests the general structure of the recursion we will use for estimating the determinant.

Fact 2.3.5. *For any matrix $\boldsymbol{M}$ where $\boldsymbol{M}_{[V_2,V_2]}$ is invertible,*

$$\det(\boldsymbol{M}_{-n}) = \det\left(\boldsymbol{M}_{[V_2,V_2]}\right) \cdot \det_+(\mathrm{Sc}(\boldsymbol{M}, V_1)).$$

This relationship also suggests that there should exist a bijection between spanning tree distribution in G and the product distribution given by sampling spanning trees independently from $\mathrm{Sc}(\boldsymbol{L}, V_1)$ and the graph Laplacian formed by adding one row/column to $\boldsymbol{L}_{[V_2,V_2]}$.

Finally, our algorithms for approximating Schur complements rely on the fact that they preserve certain marginal probabilities. The algorithms of [108, 103, 104, 88] also use variants of some of these facts, which are closely related to the preservation of the spanning tree distribution on $\mathrm{Sc}(\boldsymbol{L}, V_1)$. (See Section 2.8 for details.)

Fact 2.3.6. *Let V_1 be a subset of vertices of a graph G, then for any vertices $u, v \in V_1$, we have:*

$$\mathcal{R}^{G}_{eff}(u, v) = \mathcal{R}^{\mathrm{Sc}(G,V_1)}_{eff}(u, v).$$

Theorem 2.3.7 (Burton and Premantle [109])**.** *For any set of edges $F \subseteq E$ in a graph $G = (V, E, w)$, the probability F is contained in a w-uniform random spanning tree is*

$$\boldsymbol{Pr}^{G}_{T}(F \subseteq T) = \det(\boldsymbol{M}_{(\boldsymbol{L},F)}),$$

where $\boldsymbol{M}_{(\boldsymbol{L},F)}$ is a $|F| \times |F|$ matrix whose (e, f)'th entry, for $e, f \in F$, is given by $\sqrt{w(e)w(f)}\chi_e^T \boldsymbol{L}^{\dagger}\chi_f$.

By a standard property of Schur complements (see [110]), we have

$$\left(\boldsymbol{L}^{-1}\right)[V_1, V_1] = \mathrm{Sc}(G, V_1)^{\dagger}.$$

Here $(\boldsymbol{L}^\dagger)[V_1, V_1]$ is the minor of $\boldsymbol{L}^\dagger$ with row and column indices in V_1. This immediately implies that when F is incident only on vertices in V_1, we have $\boldsymbol{M}_{(\boldsymbol{L},F)} = \boldsymbol{M}_{(\mathrm{Sc}(G,V_1),F)}$. Putting these together, we have

Fact 2.3.8. *Given a partition of the vertices $V = V_1 \sqcup V_2$. For any set of edges F contained in $G[V_1]$, we have*

$$\boldsymbol{Pr}_T^G(F \subseteq T) = \boldsymbol{Pr}_T^{\mathrm{Sc}(G,V_1)}(F \subseteq T).$$

2.4 Sketch of the Results

The starting point for us is the paper by Janson [4] which gives (among other things) the limiting distribution of the number of spanning trees in the $\mathcal{G}_{n,m}$ model of random graphs. Our concentration result for the number of spanning trees in the sparsified graph is inspired by this paper, and our algorithmic use of this sparsification routine is motivated by sparsification based algorithms for matrices related to graphs [39, 47, 41]. The key result we will prove is a concentration bound on the number of spanning trees when the graph is sparsified by sampling edges with probability approximately proportional to effective resistance.

2.4.1 Concentration Bound

Let G be a weighted graph with n vertices and m edges, and H be a random subgraph obtained by choosing a subset of edges of size s uniformly randomly. The probability of a subset of edges, which could either be a single tree, or the union of several trees, being kept in H can be bounded precisely. Since we will eventually choose $s > n^{1.5}$, we will treat the quantity n^3/s^2 as negligible. The probability of H containing a fixed tree was shown by Janson to be:

Lemma 2.4.1. *If $m \geq \frac{s^2}{n}$, then for any tree T, the probability of it being included in H is*

$$\mathbb{P}\left[H\right] T \in H = \frac{(s)_{n-1}}{(m)_{n-1}} = p^{n-1} \cdot \exp\left(-\frac{n^2}{2s} - O\left(\frac{n^3}{s^2}\right)\right).$$

where $(a)_b$ *denotes the product* $a \cdot (a-1) \cdots (a-(b-1))$.

By linearity of expectation, the expected total weight of spanning trees in H is:

$$\mathbb{E}[H]\,\mathcal{T}_H = \mathcal{T}_G \cdot p^{n-1} \cdot \exp\left(-\frac{n^2}{2s} - O\left(\frac{n^3}{s^2}\right)\right). \tag{2.1}$$

As in [4], $\mathbb{E}[H]\,\mathcal{T}_H^2 = \mathbb{E}[H] \sum_{(T_1,T_2)} w(T_1)w(T_2)\boldsymbol{Pr}(T_1, T_2 \in H)$, can be written as a sum over all pairs of trees (T_1, T_2). Due to symmetry, the probability of a particular pair of trees T_1, T_2 both being subgraphs of H depends only on the size of their intersection. The following bound is shown in Appendix 2.11.

Lemma 2.4.2. *Let* G *be a graph with* n *vertices and* m *edges, and* H *be a uniformly random subset of* $s > 10n$ *edges chosen from* G*, where* $m \geq \frac{s^2}{n}$*. Then for any two spanning trees* T_1 *and* T_2 *of* G *with* $|T_1 \cap T_2| = k$*, we have:*

$$\mathbb{P}[H]\,T_1, T_2 \in H \leq p^{2n-2} \exp\left(-\frac{2n^2}{s}\right)\left(\frac{1}{p}\left(1 + \frac{2n}{s}\right)\right)^k,$$

where $p = s/m$.

The crux of the bound on the second moment in Janson's proof is getting a handle on the number of tree pairs (T_1, T_2) with $|T_1 \cap T_2| = k$ in the complete graph where all edges are symmetric. An alternate way to obtain a bound on the number of spanning trees can also be obtained using leverage scores, which describe the fraction of spanning trees that utilize a single edge. A well known fact about random spanning tree distributions [109] is that the edges are negatively correlated:

Fact 2.4.3 (Negative Correlation)**.** *Suppose* F *is subset of edges in a graph* G*, then*

$$\boldsymbol{Pr}_T^G(F \subseteq T) \leq \Pi_{e \in F} \boldsymbol{Pr}_T^G(e \in T).$$

An easy consequence of Fact 2.4.3 is

Lemma 2.4.4. *For any subset of edges F we have that the total weight of all spanning trees containing F is given by*

$$\sum_{\substack{T \text{ is a spanning tree of } G \\ F \subseteq T}} w(T) \leq \mathcal{T}_G \prod_{e \in F} \overline{\boldsymbol{\tau}}_e.$$

The combinatorial view of all edges being interchangable in the complete graph can therefore be replaced with an algebraic view in terms of the leverage scores. Specifically, invoking Lemma 2.4.4 in the case where all edges have leverage score at most $\frac{n}{m}$ gives the following lemma which is proven in Appendix 2.11.

Lemma 2.4.5. *In a graph G where all edges have leverage scores at most $\frac{n}{m}$, we have*

$$\sum_{\substack{T_1, T_2 \\ |T_1 \cap T_2| = k}} w(T_1) \cdot w(T_2) \leq \mathcal{T}_G^2 \cdot \frac{1}{k!} \left(\frac{n^2}{m}\right)^k$$

With Lemma 2.4.5, we can finally prove the following bound on the second moment which gives our concentration result.

Lemma 2.4.6. *Let G be a graph on n vertices and m edges such that all edges have statistical leverage scores $\leq \frac{n}{m}$. For a random subset of $s > 10n$ edges, H, where $m \geq \frac{s^2}{n}$ we have:*

$$\mathbb{E}[H] \mathcal{T}_H^2 \leq \mathcal{T}_G^2 p^{2n-2} \exp\left(-\frac{n^2}{s} + O\left(\frac{n^3}{s^2}\right)\right) = \mathbb{E}[H] \mathcal{T}_H{}^2 \exp\left(O\left(\frac{n^3}{s^2}\right)\right).$$

Proof. By definition of the second moment, we have:

$$\mathbb{E}[H] \mathcal{T}_H^2 = \sum_{T_1, T_2} w(T_1) \cdot w(T_2) \cdot \mathbb{P}[H] T_1 \cup T_2 \subseteq H.$$

Re-writing the above sum in terms of the size of the intersection k, and invoking Lemma 2.4.2

gives:

$$\mathbb{E}\left[H\right]\mathcal{T}_H^2 \leq \sum_{k=0}^{n-1} \sum_{\substack{T_1,T_2 \\ |T_1 \cap T_2|=k}} w\left(T_1\right) \cdot w\left(T_2\right) \cdot p^{2n-2} \exp\left(-\frac{2n^2}{s}\right) \left(\frac{1}{p}\left(1+\frac{2n}{s}\right)\right)^k.$$

Note that the trailing term only depends on k and can be pulled outside the summation of T_1, T_2, so we then use Lemma 2.4.5 to bound this by:

$$\mathbb{E}\left[H\right]\mathcal{T}_H^2 \leq \sum_{k=0}^{n-1} \mathcal{T}_G^2 \cdot \frac{1}{k!}\left(\frac{n^2}{m}\right)^k \cdot p^{2n-2} \exp\left(-\frac{2n^2}{s}\right) \left(\frac{1}{p}\left(1+\frac{2n}{s}\right)\right)^k.$$

Which upon pulling out the terms that are independent of k, and substituting in $p = s/m$ gives:

$$\mathbb{E}\left[H\right]\mathcal{T}_H^2 \leq \mathcal{T}_G^2 \cdot p^{2n-2} \cdot \exp\left(-\frac{2n^2}{s}\right) \cdot \sum_{k=0}^{n-1} \frac{1}{k!} \cdot \left(\frac{n^2}{s}\left(1+\frac{2n}{s}\right)\right)^k.$$

From the Taylor expansion of $\exp(\cdot)$, we have:

$$\begin{aligned}\mathbb{E}\left[H\right]\mathcal{T}_H^2 &\leq \mathcal{T}_G^2 \cdot p^{2n-2} \cdot \exp\left(-\frac{2n^2}{s}\right) \cdot \exp\left(\frac{n^2}{s}\left(1+\frac{2n}{s}\right)\right) \\ &= \mathcal{T}_G^2 \cdot p^{2n-2} \cdot \exp\left(-\frac{n^2}{s}\right) \cdot \exp\left(O\left(\frac{n^3}{s^2}\right)\right).\end{aligned}$$

□

This bound implies that once we set $s^2 > n^3$, the variance becomes less than the square of the expectation. It forms the basis of our key concentration results, which we show in Section 2.5, and also leads to Theorem 2.2.1. In particular, we demonstrate that this sampling scheme extends to importance sampling, where edges are picked with probabilities proportional to (approximations of) of their leverage scores.

A somewhat surprising aspect of this concentration result is that there is a difference between models $\mathcal{G}_{n,m}$ and the Erdos-Renyi model $\mathcal{G}_{n,p}$ when the quantity of interest is the

number of spanning trees. In particular, the number of spanning trees of a graph $G \sim \mathcal{G}_{n,m}$ is approximately normally distributed when $m = \omega\left(n^{1.5}\right)$, whereas it has approximate log-normal distribution when $G \sim \mathcal{G}_{n,p}$ and $p < 1$.

An immediate consequence of this is that we can now approximate $\det_+(\boldsymbol{L}^G)$ by computing $\det_+(\boldsymbol{L}^H)$. It also becomes natural to consider speedups of random spanning tree sampling algorithms that generate a spanning tree from a sparsifier. Note however that we cannot hope to preserve the distribution over all spanning trees via a single sparsifier, as some of the edges are no longer present.

To account for this change in support, we instead consider the randomness used in generating the sparsifier as also part of the randomness needed to produce spanning trees. In Section 2.10.1, we show that just bounds on the variance of $\mathcal{T}_H$ suffices for a bound on the TV distances of the trees.

Lemma 2.4.7. *Suppose $\mathcal{H}$ is a distribution over rescaled subgraphs of G such that for some parameter some $0 < \delta < 1$ we have*

$$\frac{\mathbb{E}\left[H \sim \mathcal{H}\right] \mathcal{T}_H^2}{\mathbb{E}\left[H \sim \mathcal{H}\right] {\mathcal{T}_H}^2} \leq 1 + \delta,$$

and for any tree $\hat{T}$ and any graph from the distribution that contain it, H we have:

$$w^H\left(\widehat{T}\right) = w^G\left(\widehat{T}\right) \cdot \mathbb{P}\left[H' \sim \mathcal{H}\right] \widehat{T} \subseteq {H'}^{-1} \cdot \frac{\mathbb{E}\left[H' \sim \mathcal{H}\right] \mathcal{T}_{H'}}{\mathcal{T}_G},$$

then the distribution given by $\boldsymbol{Pr}^G(T)$, p, and the distribution induced by $\mathbb{E}\left[H \sim \mathcal{H}\right] \boldsymbol{Pr}^H(T)$, $\tilde{p}$ satisfies

$$d_{TV}\left(p, \tilde{p}\right) \leq \sqrt{\delta}.$$

Note that uniform sampling meets the property about $w^H(T)$ because of linearity of expectation. We can also check that the importance sampling based routine that we will discuss in Section 2.5.2 also meets this criteria. Combining this with the running time

bounds from Theorem 2.2.1, as well as the $\tilde{O}(m^{1/3}n^{5/3})$ time random spanning tree sampling algorithm from [88] then leads to a faster algorithm.

Corollary 2.4.8. *For any graph G on n vertices and any $\delta > 0$, there is an algorithm that generates a tree from a distribution whose total variation is at most δ from the random tree distribution of G in time $\tilde{O}(n^{\frac{13}{6}=2.1666\ldots}\delta^{-2/3} + n^2\delta^{-2})$.*

2.4.2 Integration Into Recursive Algorithms

As a one-step invocation of our concentration bound leads to speedups over previous routines, we investigate tighter integrations of the sparsification routine into algorithms. In particular, the sparsified Schur complement algorithms [41] provide a natural place to substitute spectral sparsifiers with determinant-preserving ones. In particular, the identity of

$$\det_+(\boldsymbol{L}) = \det(\boldsymbol{L}_{[V_2,V_2]}) \cdot \det_+(\textsc{Sc}(\boldsymbol{L}, V_1)).$$

where $\det_+$ is the determinant of the matrix minor, suggests that we can approximate $\det(\boldsymbol{L}_{-n})$ by approximating $\det(\boldsymbol{L}_{[V_2,V_2]})$ and $\det_+(\textsc{Sc}(\boldsymbol{L}, V_1))$ instead. Both of these subproblems are smaller by a constant factor, and we also have $|V_1| + |V_2| = n$. So this leads to a recursive scheme where the total number of vertices involved at all layers is $O(n \log n)$. This type of recursion underlies both our determinant estimation and spanning tree sampling algorithms.

The main difficulty remaining for the determinant estimation algorithm is then sparsifying $\textsc{Sc}(G, V_1)$ while preserving its determinant. For this, we note that some V_1 are significantly easier than others: in particular, when $V_2 = V \setminus V_1$ is an independent set, the Schur complement of each of the vertices in V_2 can be computed independently. Furthermore, it is well understood how to sample these complements, which are weighted cliques, by a distribution that exceeds their true leverage scores.

Lemma 2.4.9. *There is a procedure that takes a graph G with n vertices, a parameter δ,*

and produces in $\tilde{O}(n^2\delta^{-1})$ time a subset of vertices V_1 with $|V_1| = \Theta(n)$, along with a graph H^{V_1} such that

$$\mathcal{T}_{\mathrm{Sc}(G,V_1)} \exp(-\delta) \leq \mathbb{E}\left[H^{V_1}\right] \mathcal{T}_{H^{V_1}} \leq \mathcal{T}_{\mathrm{Sc}(G,V_1)} \exp(\delta),$$

and

$$\frac{\mathbb{E}\left[H^{V_1}\right] \mathcal{T}_{H^{V_1}}^2}{\mathbb{E}\left[H^{V_1}\right] \mathcal{T}_{H^{V_1}}{}^2} \leq \exp(\delta).$$

Lemma 2.3.4 holds w.h.p., and we condition on this event. In our algorithmic applications we will be able to add the polynomially small failure probability of Lemma 2.3.4 to the error bounds.

The bound on variance implies that the number of spanning trees is concentrated close to its expectation, $\mathcal{T}_{\mathrm{Sc}(G,V_1)}$, and that a random spanning tree drawn from the generated graph H^{V_1} is —over the randomness of the sparsification procedure—close in total variation distance to a random spanning tree of the true Schur complement.

As a result, we can design schemes that:

1. Finds an $O(1)$-DD subset V_2, and set $V_1 \leftarrow V \setminus V_2$.
2. Produce a determinant-preserving sparsifier H^{V_1} for $\mathrm{Sc}(G, V_1)$.
3. Recurse on both $\boldsymbol{L}_{[V_2,V_2]}$ and H^{V_1}.

However, in this case, the accumulation of error is too rapid for yielding a good approximation of determinants. Instead, it becomes necesary to track the accumulation of variance during all recursive calls. Formally, the cost of sparsifying so that the variance is at most δ is about $n^2\delta^{-1}$, where δ is the size of the problem. This means that for a problem on G_i of size $\beta_i n$ for $0 \leq \beta_i \leq 1$, we can afford an error of $\beta_i\delta$ when working with it, since:

1. The sum of β_i on any layer is at most 2, [5] so the sum of variance per layer is $O(\delta)$.

[5] each recursive call may introduce one new vertex

2. The cost of each sparsification step is now $\beta_i n^2 \delta^{-1}$, which sums to about $n^2\delta^{-1}$ per layer.

Our random spanning tree sampling algorithm in Section 2.8 is similarly based on this careful accounting of variance. We first modify the recursive Schur complement algorithm introduced by Coulburn et al. [108] to give a simpler algorithm that only braches two ways at each step in Section 2.8.1, leading to a high level scheme fairly similar to the recursive determinant algorithm. Despite these similarities, the accumulation of errors becomes far more involved here due to the choice of trees in earlier recursive calls affecting the graph in later steps. More specifically, the recursive structure of our determinant algorithm can be considered analogous to a breadth-first-search, which allows us to consider all subgraphs at each layer to be independent. In contrast, the recursive structure of our random spanning tree algorithm, which we show in Section 2.8.2 is more analogous to a depth-first traversal of the tree, where the output solution of one subproblem will affect the input of all subsequent subproblems.

These dependency issues will be the key difficulty in considering variance across levels. The total variation distance tracks the discrepancy over all trees of G between their probability of being returned by the overall recursive algorithm, and their probability in the w-uniform distribution. Accounting for this over all trees leads us to bounding variances in the probabilities of individual trees being picked. As this is, in turn, is equivalent to the weight of the tree divided by the determinant of the graph, the inverse of the probability of a tree being picked can play a simliar role to the determinant in the determinant sparsification algorithm described above. However, tracking this value requires analyzing extending our concentration bounds to the case where an arbitrary tree is fixed in the graph and we sample from the remaining edges. We study this Section 2.9, prove bounds analogous to the concentration bounds from Section 2.5, and incorporate the guarantees back into the recursive algorithm in Section 2.8.2.

2.5 Determinant Preserving Sparsification

In this section we will ultimately prove Theorem 2.2.1, our primary result regarding determinant-preserving sparsification. However, most of this section will be devoted to proving the following general determinant-preserving sparsification routine that also forms the core of subsequent algorithms:

Theorem 2.5.1. *Given an undirected, weighted graph $G = (V, E, w)$, an error threshold $\epsilon > 0$, parameter ρ along with routines:*

1. SAMPLEEDGE$_G()$ *that samples an edge e from a probability distribution $\boldsymbol{p}$ ($\sum_e \boldsymbol{p}_e = 1$), as well as returning the corresponding value of $\boldsymbol{p}_e$. Here $\boldsymbol{p}_e$ must satisfy:*

$$\frac{\overline{\boldsymbol{\tau}}_e}{n-1} \leq \rho \cdot \boldsymbol{p}_e$$

   where $\overline{\boldsymbol{\tau}}_e$ is the true leverage score of e in G.

2. APPROXLEVERAGE$_G(u, v, \epsilon)$ *that returns the leverage score of an edge u, v in G to an error of ϵ. Specifically, given an edge e, it returns a value $\widetilde{\boldsymbol{\tau}}_e$ such that:*

$$(1-\epsilon)\,\overline{\boldsymbol{\tau}}_e \leq \widetilde{\boldsymbol{\tau}}_e \leq (1+\epsilon)\,\overline{\boldsymbol{\tau}}_e.$$

There is a routine DETSPARSIFY(G, s, ϵ) *that computes a graph H with s edges such that its tree count, $\mathcal{T}_H$, satisfies:*

$$\mathbb{E}\left[H\right]\mathcal{T}_H = \mathcal{T}_G\left(1 \pm O\left(\frac{n^3}{s^2}\right)\right),$$

and:

$$\frac{\mathbb{E}\left[H\right]\mathcal{T}_H^2}{\mathbb{E}\left[H\right]\mathcal{T}_H{}^2} \leq \exp\left(\frac{\epsilon^2 n^2}{s} + O\left(\frac{n^3}{s^2}\right)\right)$$

Furthermore, the expected running time is bounded by:

1. $O(s \cdot \rho)$ *calls to* SAMPLEEDGE$_G(e)$ *and* APPROXLEVERAGE(e) *with constant error,*

2. $O(s)$ *calls to* APPROXLEVERAGE(e) *with* ϵ *error.*

We establish guarantees for this algorithm using the following steps:

1. Showing that the concentration bounds as sketched in Section 2.4 holds for approximate leverage scores in Section 2.5.1.

2. Show via taking the limit of probabilistic processes that the analog of this process works for sampling a general graph where edges can have varying leverage scores. This proof is in Section 2.5.2.

3. Show via rejection sampling that (high error) one sided bounds on statistical leverage scores, such as those that suffice for spectral sparsification, can also be to do the initial round of sampling instead of two-sided approximations of leverage scores. This, as well as pseudocode and guarantees of the overall algorithm are given in Section 2.5.3.

2.5.1 Concentration Bound with Approximately Uniform Leverage Scores

Similar to the simplified proof as outlined in Section 2.4, our proofs relied on uniformly sampling s edges from a multi-graph with $m \geq \frac{s^2}{n}$ edges, such that all edges have leverage score within multiplicative $1 \pm \epsilon$ of $\frac{n-1}{s}$, aka. approximately uniform. The bound that we prove is an analog of Lemma 2.4.6

Lemma 2.5.2. *Given a weighted multi-graph* G *such that* $m \geq \frac{s^2}{n}$, $s \geq n$, *and all edges* $e \in E$ *have* $\frac{(1-\epsilon)(n-1)}{m} \leq \overline{\boldsymbol{\tau}}_e \leq \frac{(1+\epsilon)(n-1)}{m}$, *with* $0 \leq \epsilon < 1$, *then*

$$\frac{\mathbb{E}\left[H\right]\mathcal{T}_H^2}{\mathbb{E}\left[H\right]\mathcal{T}_H{}^2} \leq \exp\left(\frac{n^2\epsilon^2}{s} + O\left(\frac{n^3}{s^2}\right)\right)$$

Similar to the proof of Lemma 2.4.6 in Section 2.4, we can utilize the bounds on the probability of k edges being chosen using Lemma 2.4.2. The only assumption that changed

was the bounds on $\overline{\boldsymbol{\tau}}_e$, which does not affect $\mathbb{E}[H]\,{\mathcal{T}_H}^2$. The only term that changes is our upper bound the total weight of trees that contain some subset of k edges that was the produce of k leverage scores. At a glance, this product can change by a factor of up to $(1+\epsilon)^k$, which when substituted naively into the proof of Lemma 2.4.2 directly would yield an additional term of

$$\exp\left(\frac{n^2\epsilon}{s}\right),$$

and in turn necessitating $\epsilon < n^{-1/2}$ for a sample count of $s \approx n^{1.5}$.

However, note that this is the worst case distortion over a subset F. The upper bound that we use, Lemma 2.4.5 sums over these bounds over all subsets, and over all edges we still have

$$\sum e \in G\overline{\boldsymbol{\tau}}_e = n-1.$$

Incorporating this allows us to show a tighter bound that depends on ϵ^2.

Similar to the proof of Lemma 2.4.5, we can regroup the summation over all $\binom{m}{k}$ subsets of $E(G)$, and bound the fraction of trees containing each subset F via $\sum_{T:F\subseteq T} w(T) \leq \mathcal{T}_G \prod_{e\in F} \boldsymbol{\tau}_e$ via Lemma 2.4.4.

$$\sum_{\substack{T_1,T_2\\|T_1\cap T_2|=k}} w(T_1)\cdot w(T_2) \leq \sum_{\substack{F\subseteq E\\|F|=k}} \mathcal{T}_G^2 \prod_{e\in F} \overline{\boldsymbol{\tau}}_e^2$$

The proof will heavily utilize the fact that $\sum_{e\in E} \overline{\boldsymbol{\tau}}_e = n-1$. We bound this in first two steps: first treat it as a symmetric product over $\overline{\boldsymbol{\tau}}_e^2$, and bound the total as a function of

$$\sum_e \overline{\boldsymbol{\tau}}_e^2,$$

then we bound this sum using the fact that $\sum_e \overline{\boldsymbol{\tau}}_e = n-1$.

The first step utilizes the concavity of the product function, and bound the total by the sum:

Lemma 2.5.3. *For any set of non-negative values $\mathbf{x}_1 \dots \mathbf{x}_m$ with $\sum_i \mathbf{x}_i \leq z$, we have*

$$\sum_{\substack{F \subseteq [1 \dots m] \\ |F|=k}} \prod_{i \in F} \mathbf{x}_i \leq \binom{m}{k} \left(\frac{z}{m}\right)^k.$$

Proof. We claim that this sum is maximized when $\mathbf{x}_i = \left(\frac{z}{m}\right)$ for all e.

Consider fixing all variables other than some $\mathbf{x}_i$ and $\mathbf{x}_j$, which we assume to be $\mathbf{x}_1 \leq \mathbf{x}_2$ without loss of generality as the function is symmetric on all variables:

$$\sum_{\substack{F \subseteq [1 \dots m] \\ |F|=k}} \prod_{i \in F} \mathbf{x}_i = \mathbf{x}_1 \mathbf{x}_2 \left(\sum_{\substack{F \subseteq [3 \dots m] \\ |F|=k-2}} \prod_{i \in F} \mathbf{x}_i \right) + (\mathbf{x}_1 + \mathbf{x}_2) \cdot \left(\sum_{\substack{F \subseteq [3 \dots m] \\ |F|=k-1}} \prod_{i \in F} \mathbf{x}_i \right) + \sum_{\substack{F \subseteq [3 \dots m] \\ |F|=k}} \prod_{i \in F} \mathbf{x}_i.$$

Then if $\mathbf{x}_1 < \mathbf{x}_2$, locally changing their values to $\mathbf{x}_1 + \epsilon$ and $\mathbf{x}_2 - \epsilon$ keeps the second term the same. While the first term becomes

$$(\mathbf{x}_1 + \epsilon)(\mathbf{x}_2 - \epsilon) = \mathbf{x}_1 \mathbf{x}_2 + \epsilon(\mathbf{x}_2 - \mathbf{x}_1) - \epsilon^2,$$

which is greater than $\mathbf{x}_1 \mathbf{x}_2$ when $0 < \epsilon < (\mathbf{x}_2 - \mathbf{x}_1)$.

This shows that the overall summation is maximized when all $\mathbf{x}_i$ are equal, aka

$$\mathbf{x}_i = \frac{z}{m},$$

which upon substitution gives the result. □

The second step is in fact the $k = 1$ case of Lemma 2.4.5.

Lemma 2.5.4. *For any set of values $\mathbf{y}_e$ such that*

$$\sum_e \mathbf{y} = n - 1,$$

and

$$\frac{(1-\epsilon)\,n}{m} \leq \mathbf{y}_e \leq \frac{(1+\epsilon)\,n}{m},$$

we have

$$\sum_e \mathbf{y}_e^2 \leq \frac{(1+\epsilon^2)(n-1)^2}{m}.$$

Proof. Note that for any $a \leq b$, and any ϵ, we have

$$(a-\epsilon)^2 + (b+\epsilon)^2 = a^2 + b^2 + 2\epsilon^2 + 2\epsilon\,(b-a)\,,$$

and this transformation must increase the sum for $\epsilon > 0$. This means the sum is maximized when half of the leverage scores are $\frac{(1-\epsilon)(n-1)}{m}$ and the other half are $\frac{(1+\epsilon)(n-1)}{m}$. This then gives

$$\sum_{e\in E} \mathbf{y}_e^2 \leq \frac{m}{2}\left(\frac{(1+\epsilon)(n-1)}{m}\right)^2 + \frac{m}{2}\left(\frac{(1-\epsilon)(n-1)}{m}\right)^2 = \frac{(1+\epsilon^2)(n-1)^2}{m}.$$

□

Proof. (of Lemma 2.5.2)

We first derive an analog of Lemma 2.4.5 for bounding the total weights of pairs of trees containing subsets of size k, where we again start with the bounds

$$\sum_{\substack{T_1,T_2\\ |T_1\cap T_2|=k}} w\,(T_1)\cdot w\,(T_2) \leq \sum_{\substack{F\subseteq E\\ |F|=k}} \sum_{\substack{T_1,T_2\\ F\subseteq T_1\cap T_2}} w\,(T_1)\cdot w\,(T_2) = \sum_{\substack{F\subseteq E\\ |F|=k}} \left(\sum_{T:F\subseteq T} w\,(T)\right)^2$$

Applying Lemma 2.4.4 to the inner term of the summation then gives

$$\sum_{\substack{T_1,T_2\\ |T_1\cap T_2|=k}} w\,(T_1)\cdot w\,(T_2) \leq \sum_{\substack{F\subseteq E\\ |F|=k}} \mathcal{T}_G^2 \cdot \prod_{e\in F} \overline{\boldsymbol{\tau}}_e^2$$

The bounds on $\overline{\boldsymbol{\tau}}_e$ and $\sum_e \overline{\boldsymbol{\tau}}_e = n-1$ gives, via Lemma 2.5.4

$$\sum_e \overline{\boldsymbol{\tau}}_e^2 \leq \frac{(1+\epsilon^2)(n-1)^2}{m}.$$

Substituting this into Lemma 2.5.3 with $\mathbf{x}_i = \overline{\boldsymbol{\tau}}_e^2$ then gives

$$\sum_{\substack{F \subseteq E \\ |F|=k}} \prod_{e \in F} \overline{\boldsymbol{\tau}}_e^2 \leq \binom{m}{k} \left(\frac{(1+\epsilon^2)n^2}{m^2}\right)^k \leq \frac{m^k}{k!} \left(\frac{(1+\epsilon^2)n^2}{m^2}\right)^k = \frac{1}{k!} \left(\frac{(1+\epsilon^2)n^2}{m}\right)^k.$$

which implies our analog of Lemma 2.4.5

$$\sum_{\substack{T_1, T_2 \\ |T_1 \cap T_2| = k}} w(T_1) \cdot w(T_2) \leq \mathcal{T}_G^2 \cdot \frac{1}{k!} \left(\frac{(1+\epsilon^2)n^2}{m}\right)^k.$$

We can then duplicate the proof of Lemma 2.4.6. Similar to that proof, we can regroup the summation by $k = |T_1 \cap T_2|$ and invoking Lemma 2.4.2 to get:

$$\mathbb{E}[H]\,\mathcal{T}_H^2 \leq \sum_{k=0}^{n-1} \sum_{\substack{T_1, T_2 \\ |T_1 \cap T_2| = k}} w(T_1) \cdot w(T_2) \cdot p^{2n-2} \exp\left(-\frac{2n^2}{s}\right) \left(\frac{1}{p}\left(1 + \frac{2n}{s}\right)\right)^k.$$

where $p = s/m$. When incorporated with our analog of Lemma 2.4.5 gives:

$$\begin{aligned}\mathbb{E}[H]\,\mathcal{T}_H^2 &\leq \sum_{k=0}^{n-1} p^{2n-2} \exp\left(-\frac{2n^2}{s}\right) \left(\frac{1}{p}\left(1 + \frac{2n}{s}\right)\right)^k \cdot \mathcal{T}_G^2 \frac{1}{k!} \left(\frac{(1+\epsilon^2)n^2}{m}\right)^k \\ &= \mathcal{T}_G^2 p^{2n-2} \cdot \exp\left(-\frac{2n^2}{s}\right) \cdot \sum_{k=0}^{n-1} \frac{1}{k!} \cdot \left(\frac{(1+\epsilon^2)n^2}{s}\left(1 + \frac{2n}{s}\right)\right)^k.\end{aligned}$$

Substituting in the Taylor expansion of $\sum_k \frac{z^k}{k!} \leq \exp(z)$ then leaves us with:

$$\mathbb{E}[H]\,\mathcal{T}_H^2 \leq \mathcal{T}_G^2 \cdot p^{2n-2} \cdot \exp\left(-\frac{n^2}{s} + \frac{n^2\epsilon^2}{s} + O\left(\frac{n^3}{s^2}\right)\right)$$

and finishes the proof.

□

2.5.2 Generalization to Graphs with Arbitrary Leverage Score Distributions

The first condition of $m \geq \frac{s^2}{n}$ will be easily achieved by splitting each edge a sufficient number of times, which does not need to be done explicitly in the sparsification algorithm. Furthermore, from the definition of statistical leverage score splitting an edge into k copies will give each copy a kth fraction of the edge's leverage score. Careful splitting can then ensure the second condition, but will require ϵ-approximate leverage score estimates on the edges. The simple approach would compute this for all edges, then split each edge according to this estimate and draw from the resulting edge set. Instead, we only utilize this algorithm as a proof technique, and give a sampling scheme that's equivalent to this algorithm's limiting behavior as $m \to \infty$. Pseudocode of this routine is in Algorithm 1.

Algorithm 1: IDEALSPARSIFY$(G, \widetilde{\boldsymbol{\tau}}, s)$: Sample s (multi) edges of G to produce H such that $\mathcal{T}_G \approx \mathcal{T}_H$.

Input: Graph G, approximate leverage scores $\widetilde{\boldsymbol{\tau}}$, sample count s

Initialize H as the empty graph, $H \leftarrow \emptyset$;

for $i = 1 \ldots s$ **do**

  Pick edge e with probability proportional to $\widetilde{\boldsymbol{\tau}}_e$;

  Add e to H with new weight:

$$\frac{w_e (n-1)}{\widetilde{\boldsymbol{\tau}}_e s} \exp\left(\frac{n^2}{2(n-1)s}\right).$$

Output H

Note that this sampling scheme is with replacement: the probability of a 'collision' as the number of copies tend to ∞ is sufficiently small that it can be covered by the proof as well.

The guarantee that we will show for Algorithm 1 is:

Lemma 2.5.5. *For any graph G and any set of approximate leverage scores $\widetilde{\boldsymbol{\tau}}$ such that*

$$(1-\epsilon)\,\boldsymbol{\tau}_e \leq \widetilde{\boldsymbol{\tau}}_e \leq (1+\epsilon)\,\boldsymbol{\tau}_e$$

for all edges e. The graph $H = \text{IDEALSPARSIFY}(G, \widetilde{\boldsymbol{\tau}}, s)$ satisfies:

$$\left(1 - O\left(\frac{n^3}{s^2}\right)\right) \mathcal{T}_G \leq \mathbb{E}\left[H\right] \mathcal{T}_H \leq \mathcal{T}_G,$$

and

$$\frac{\mathbb{E}\left[H\right] \mathcal{T}_H^2}{\mathbb{E}\left[H\right] {\mathcal{T}_H}^2} \leq \exp\left(O\left(\frac{\epsilon^2 n^2}{s} + \frac{n^3}{s^2}\right)\right).$$

Our proof strategy is simple: claim that this algorithm is statistically close to simulating splitting each edge into a very large number of copies. Note that these proofs are purely for showing the convergence of statistical processes, so all that's needed is for the numbers that arise in this proof (in particular, m) to be finite.

We first show that G and $\widetilde{\boldsymbol{\tau}}$ can be perturbed to become rational numbers.

Lemma 2.5.6. *For any graph G and any set of $\widetilde{\boldsymbol{\tau}}$ such that $(1-\epsilon)\overline{\boldsymbol{\tau}}_e^{(G)} \leq \widetilde{\boldsymbol{\tau}}_e \leq (1+\epsilon)\overline{\boldsymbol{\tau}}_e^{(G)}$ for all edges e for some constant $\epsilon > 0$, and any perturbation threshold δ, we can find graph G' with all edge weights rationals, and $\widetilde{\boldsymbol{\tau}}'$ with all entries rational numbers such that:*

1. *$\mathcal{T}_G \leq \mathcal{T}_{G'} \leq (1+\delta)\mathcal{T}_G$, and*
2. *$(1-2\epsilon)\overline{\boldsymbol{\tau}}_e^{(G')} \leq \widetilde{\boldsymbol{\tau}}'_e \leq (1+2\epsilon)\overline{\boldsymbol{\tau}}_e^{(G')}$ for all edges e.*

Proof. This is a direct consequence of the rational numbers being everywhere dense, and that perturbing edge weights by a factor of $1 \pm \alpha$ perturbs leverage scores by a factor of up to $1 \pm O(\alpha)$, and total weights of trees by a factor of $(1 \pm \alpha)^{n-1}$. □

Having all leverage scores as integers means that we can do an exact splitting by setting m, the total number of split edges, to a multiple of the common denominator of all the $\widetilde{\boldsymbol{\tau}}'_e$ values times $n-1$. Specifically, an edge with approximate leverage score $\widetilde{\boldsymbol{\tau}}'_e$ becomes

$$\widetilde{\boldsymbol{\tau}}'_e \cdot \frac{m}{n-1}$$

copies, each with weight

$$\frac{w_e\left(n-1\right)}{\widetilde{\boldsymbol{\tau}}'_e m},$$

and 'true' leverage score

$$\frac{\overline{\boldsymbol{\tau}}_e\left(n-1\right)}{\widetilde{\boldsymbol{\tau}}'_e m}.$$

In particular, since

$$\left(1-2\epsilon\right) \leq \frac{\overline{\boldsymbol{\tau}}_e}{\widetilde{\boldsymbol{\tau}}_e} \leq \left(1+2\epsilon\right),$$

this splitted graph satisfies the condition of Lemma 2.5.2. This then enables us to obtain the guarantees of Lemma 2.5.5 by once again letting m tend to ∞.

Proof. (of Lemma 2.5.5) We first show that Algorithm 1 works for the graph with rational weights and approximate leverage scores as generated by Lemma 3.4.3.

The condition established above means that we can apply Lemma 2.5.2 to the output of picking s random edges among these m split copies. This graph H' satisfies

$$\mathbb{E}\left[H'\right]\mathcal{T}_{H'} = \mathcal{T}_{G'}\left(\frac{s}{m}\right)^{n-1}\exp\left(-\frac{n^2}{2s}-O\left(\frac{n^3}{s^2}\right)\right),$$

and

$$\frac{\mathbb{E}\left[H'\right]\mathcal{T}_{H'}^2}{\mathbb{E}\left[H'\right]\mathcal{T}_{H'}{}^2} \leq \exp\left(\frac{n^2\epsilon^2}{s}+O\left(\frac{n^3}{s^2}\right)\right).$$

The ratio of the second moment is not affected by rescaling, so the graph

$$H'' \leftarrow \frac{m}{s}\exp\left(\frac{n^2}{2s\left(n-1\right)}\right)$$

meets the requirements on both the expectation and variances. Furthermore, the rescaled weight of an single edge being picked is:

$$\frac{w_e\left(n-1\right)}{\widetilde{\boldsymbol{\tau}}'_e m}\cdot\frac{m}{s}\exp\left(\frac{n^2}{2s\left(n-1\right)}\right) = \frac{w_e\left(n-1\right)}{\widetilde{\boldsymbol{\tau}}'_e s}\exp\left(\frac{n^2}{2s\left(n-1\right)}\right),$$

which is exactly what Algorithm 1 assigns.

It remains to resolve the discrepancy between sampling with and without replacement: the probability of the same edge being picked twice in two different steps is at most $1/m$, so the total probability of a duplicate sample is bounded by s^2/m. We then give a finite bound on the size of m for which this probability becomes negligible in our routine. The rescaling factor of a single edge is (very crudely) bounded by

$$\frac{(n-1)}{\widetilde{\boldsymbol{\tau}}'_e s} \exp\left(\frac{n^2}{2s\,(n-1)}\right) \leq \exp\left(n^3\right) \frac{1}{\min_e \widetilde{\boldsymbol{\tau}}'_e},$$

which means that any of the H'' returned must satisfy

$$\mathcal{T}_{H''} \leq \exp\left(n^4\right) \left(\frac{1}{\min_e \widetilde{\boldsymbol{\tau}}'_e}\right)^n \mathcal{T}_{G'},$$

which is finite. As a result, as $m \to \infty$, the difference that this causes to both the first and second moments become negligible.

The result for $H \leftarrow$ IDEALSPARSIFY$(G, \widetilde{\boldsymbol{\tau}}, s)$ then follows from the infinitesimal perturbation made to G, as the rational numbers are dense everywhere. □

2.5.3 Incorporating Crude Edge Sampler Using Rejection Sampling

Under Lemma 2.5.5 we assumed access to ϵ-approximate leverage scores, which could be computed with m calls to our assumed subroutine APPROXLEVERAGE$_G$, where m here is the number of edges of G. However, we roughly associate APPROXLEVERAGE$_G$ with Lemma 2.3.4 that requires $\tilde{O}(\epsilon^{-2})$ time per call (and we deal with the w.h.p. aspect in the proof of Theorem 2.2.1), and to achieve our desired sparsification of $O(n^{1.5})$ edges, we will need $\epsilon = n^{-1/4}$ for the necessary concentration bounds. Instead, we will show that we can use rejection sampling to take s edges drawn from approximate leverage scores using a cruder distribution $\boldsymbol{p}_e$, which will only require application of APPROXLEVERAGE$_G$ with error ϵ for an expected $O(s)$ number of edges.

Rejection sampling is a known technique that allows us to sample from some distribution f by instead sampling from a distribution g that approximates f and accept the sample with a specific probability based on the probability of drawing that sample from f and g.

More specifically, suppose we are given two probability distributions f and g over the same state space X, such that for all $x \in X$ we have $Cg(x) \geq f(x)$ for some constant C. Then we can draw from f by instead drawing $x \sim g$, and accepting the draw with probability $\frac{f(x)}{Cg(x)}$.

This procedure only requires a lower bound on g with respect to f, but in order to accept a draw with constant probability, there need to be weaker upper bound guarantees. Our guarantees on $\widetilde{\boldsymbol{\tau}}_e$ will fulfill these requirements, and the rejection sampling will accept a constant fraction of the draws. By splitting into a sufficient number of edges, we ensure that drawing the same multi-edge from any split edge will occur with at most constant probability.

Specifically, each sample is drawn via. the following steps:

1. Draw a sample according the distribution g, e.
2. Evaluate the values of $f(e)$ and $g(e)$.
3. Keep the sample with probability $f(e)/g(e)$.

As the running time of ApproxLeverage$_G(e, \epsilon)$ will ultimately depend on the value of ϵ apply this algorithmic framework, we also need to perform rejection sampling twice, once with constant error, and once with leverage scores extracted from the true approximate distribution. Pseudocode of this routine is shown in Algorithm 2.

We first show that this routine will in fact sample edges according to ϵ-approximate leverage scores, as was assumed in IdealSparsify

Lemma 2.5.7. *The edges are being sampled with probability proportional to $\widetilde{\boldsymbol{\tau}}^{(G,\epsilon)}$, the leverage score estimates given by* ApproxLeverage$_G(\cdot, \epsilon)$.

Algorithm 2: DETSPARSIFY$(G, s, \text{SAMPLEEDGE}_G(), \rho, \text{APPROXLEVERAGE}_G(u, v, \epsilon))$: Sample s (multi) edges of G to produce H such that $\mathcal{T}_G \approx \mathcal{T}_H$.

Input: Graph G.
Sample count s, leverage score approximation error $0 < \epsilon < 1/2$,
$\text{SAMPLEEDGE}_G()$ that samples an edge e from a probability distribution $\boldsymbol{p}$ ($\sum_e \boldsymbol{p}_e = 1$), and returning the corresponding value of $\boldsymbol{p}_e$.
ρ that bounds the under-sampling rate of $\text{SAMPLEEDGE}_G()$.
$\text{APPROXLEVERAGE}_G(u, v, \epsilon)$ that returns the approximate leverage score of an edge u, v in G to an error of ϵ.

Initialize H as the empty graph, $H \leftarrow \emptyset$;
while *H has fewer than s edges* **do**
  $e, \boldsymbol{p}_e \leftarrow \text{SAMPLEEDGE}_G()$.
  Let $\boldsymbol{p}'_e \leftarrow \frac{2}{n-1}\text{APPROXLEVERAGE}_G(u, v, 0.1)$
  Reject e with probability $1 - \boldsymbol{p}'_e/(4\rho \cdot \boldsymbol{p}_e)$.
  Let $\boldsymbol{p}''_e \leftarrow \frac{1}{n-1}\text{APPROXLEVERAGE}_G(u, v, \epsilon)$
  Reject e with probability $1 - \boldsymbol{p}''_e/\boldsymbol{p}'_e$.
  Add e to H with new weight

$$\frac{w_e}{\boldsymbol{p}''_e s} \exp\left(\frac{n^2}{2(n-1)s}\right).$$

Output H

Note that this algorithm does not, at any time, have access to the full distribution $\widetilde{\boldsymbol{\tau}}^{(G,\epsilon)}$.

Proof. Our proof will assume the known guarantees of rejection sampling, which is to say that the following are true:

1. Given distributions $\boldsymbol{p}$ and $\boldsymbol{p}'$, sampling an edge e from $\boldsymbol{p}$ and accepting with probability $\boldsymbol{p}'_e/(4\rho\cdot\boldsymbol{p}_e)$ is equivalent to drawing an edge from $\boldsymbol{p}'$ as long as $\boldsymbol{p}'_e/(4\rho\cdot\boldsymbol{p}_e) \in [0,1]$ for all e.

2. Given distributions $\boldsymbol{p}'$ and $\boldsymbol{p}''$, sampling an edge e from $\boldsymbol{p}'$ and accepting with probability $\boldsymbol{p}''_e/\boldsymbol{p}'_e$ is equivalent to drawing an edge from $\boldsymbol{p}''$ as long as $\boldsymbol{p}''_e/\boldsymbol{p}'_e \in [0,1]$ for all e.

As a result, we only need to check that $\boldsymbol{p}'_e/(4\rho\boldsymbol{p}_e)$ and $\boldsymbol{p}''_e/\boldsymbol{p}'_e$ are at most 1.

The guarantees of $\textsc{SampleEdge}_G()$ gives

$$\frac{\overline{\boldsymbol{\tau}}_e}{n-1} \leq \rho\boldsymbol{p}_e.$$

As $\boldsymbol{p}'_e$ was generated with error 1.1, we have

$$\boldsymbol{p}'_e \leq \frac{2.2\overline{\boldsymbol{\tau}}_e}{(n-1)} \leq 2.2\rho\boldsymbol{p}_e,$$

so $\boldsymbol{p}'_e/(4\rho\boldsymbol{p}_e) \leq 1$. To show $\boldsymbol{p}''_e/\boldsymbol{p}'_e \leq 1$, once again the guarantees of $\textsc{SampleEdge}_G()$ gives:

$$\boldsymbol{p}''_e \leq (1+\epsilon)\frac{\overline{\boldsymbol{\tau}}_e}{n-1} \leq 2\cdot 0.9\frac{\overline{\boldsymbol{\tau}}_e}{n-1} \leq \boldsymbol{p}'_e.$$

□

It remains to show that this rejection sampling process still makes sufficiently progress, yet also does not call $\textsc{ApproxLeverage}_G(e,\epsilon)$ (the more accurate leverage score estimator) too many times.

Lemma 2.5.8. *The probability of* DETSPARSIFY *calling* APPROXLEVERAGE$_G(e,\epsilon)$ *is at most* $\frac{1}{\rho}$*, while the probability of it adding an edge to* H *is at least* $\frac{1}{8\rho}$.

Proof. The proof utilizes the fact $\sum_e \overline{\boldsymbol{\tau}}_e = n-1$ (Fact 2.3.3) extensively.

If the edge e is picked, APPROXLEVERAGE$_G(e,\epsilon)$ is called with probability

$$\frac{\boldsymbol{p}'_e}{4\rho \cdot \boldsymbol{p}_e} \leq \frac{2.2\overline{\boldsymbol{\tau}}_e}{4\rho \cdot \boldsymbol{p}_e \cdot (n-1)}$$

Summing over this over all edge e by the probability of picking them gives:

$$\sum_e \boldsymbol{p}_e \frac{2.2\overline{\boldsymbol{\tau}}_e}{4\rho \cdot \boldsymbol{p}_e \cdot (n-1)} = \frac{2.2\sum_e \overline{\boldsymbol{\tau}}_e}{4\rho \cdot (n-1)} \leq \frac{1}{\rho}.$$

On the other hand, the probability of picking edge e, and not rejecting it is:

$$\boldsymbol{p}_e \cdot \frac{\boldsymbol{p}'_e}{4\rho \cdot \boldsymbol{p}_e} \cdot \frac{\boldsymbol{p}''_e}{\boldsymbol{p}'_e} = \frac{\widetilde{\boldsymbol{\tau}}^{(G,\epsilon)}}{4\rho(n-1)}.$$

where this follows by cancellation and how we set $\boldsymbol{p}''_e$ in our algorithm. Summing over all edges then gives the probability of not rejecting an edge to be

$$\sum_e \frac{\widetilde{\boldsymbol{\tau}}^{(G,\epsilon)}}{4\rho(n-1)} \geq \sum_e \frac{(1-\epsilon)\overline{\boldsymbol{\tau}}_e}{4\rho(n-1)} = \frac{(1-\epsilon)\sum_e \overline{\boldsymbol{\tau}}_e}{4\rho(n-1)} \geq \frac{1}{8\rho}$$

□

Proof. (of Theorem 2.5.1) Lemma 2.5.7 implies that edges are sampled in DETSPARSIFY with probability proportional to ϵ-approximate leverage scores from APPROXLEVERAGE$_G(\cdot,\epsilon)$. Therefore, we can apply Lemma 2.5.5 to achieve the desired expectation and concentration bounds. Finally, Lemma 2.5.8 implies that we expect to sample at most $O(s \cdot \rho)$ edges, each of which require a call to SAMPLEEDGE$_G(e)$ and APPROXLEVERAGE$_G$ with constant error. It additionally implies that we expect to make $O(s)$ calls to APPROXLEVERAGE$_G$ with ϵ error.

□

Directly invoking this theorem leads to the sparsification algorithm.

Proof. (of Theorem 2.2.1) Consider invoking Theorem 2.5.1 with parameters

$$s \leftarrow O\left(n^{1.5}\delta^{-2}\right),$$
$$\epsilon \leftarrow n^{-1/4}.$$

This gives:

$$\frac{\epsilon^2 n^2}{s}, \frac{n^3}{s^2} \leq \delta,$$

which then implies

$$\left(1 - O\left(\delta^2\right)\right)\mathcal{T}_G \leq \mathbb{E}\left[H\right]\mathcal{T}_H \leq \left(1 + O\left(\delta^2\right)\right)\mathcal{T}_G,$$

and

$$\mathbb{E}\left[H\right]\mathcal{T}_H^2 \leq \left(1 + O\left(\delta^2\right)\right)\mathbb{E}\left[H\right]{\mathcal{T}_H}^2.$$

The second condition is equivalent to $\mathbf{Var}\left[H\right]\mathcal{T}_H \leq \delta^2\mathbb{E}\left[H\right]\mathcal{T}_H$, which by Chebyshev inequality gives that with constant probability we have

$$\left(1 - O\left(\delta\right)\right)\mathcal{T}_G \leq \mathcal{T}_H \leq \left(1 + O\left(\delta\right)\right)\mathcal{T}_G.$$

Combining this with the bounds on $\mathbb{E}\left[H\right]\mathcal{T}_H$, and adjusting constants gives the overall bound.

Constructing the probability distribution $\boldsymbol{p}$ for sampling edges only requires computing constant approximate leverage scores for all edges, and then sampling proportionally for each edge, giving a constant value for ρ. By Lemma 2.3.4, this requires $\tilde{O}(m)$ time. The running time then is dominated by the $O(s)$ calls made to the effective resistance oracle

with error $\epsilon = n^{-1/4}$. Invoking Lemma 2.3.4 gives that this cost is bounded by

$$O\left(n\epsilon^{-4} + s\epsilon^{-2}\right) = O\left(n^2\delta^{-2}\right).$$

Furthermore, because Lemma 2.3.4 holds w.h.p. we can absorb the probability of failure into our constant probability bound □

Another immediate consequence of our sparsification routine in Theorem 2.5.1, along with bounds on total variation distance that we prove in Section 2.10, is that we can give a faster spanning tree sampling algorithm for dense graphs by plugging the sparsified graph into previous algorithms for generating random spanning trees.

Proof. (of Corollary 2.4.8) As in the proof of Theorem 2.2.1, we invoke Theorem 2.5.1 with parameters

$$s \leftarrow O\left(n^{1.5}\delta^{-2}\right),$$
$$\epsilon \leftarrow n^{-1/4}.$$

giving

$$\frac{\mathbb{E}\left[H\right]\mathcal{T}_H^2}{\mathbb{E}\left[H\right]{\mathcal{T}_H}^2} \leq 1 + \delta^2.$$

Applying Lemma 2.4.7, which is proven in Section 2.10.1, we then have that drawing a tree from H according to the w-uniform distribution gives a total variation distance of δ from drawing a tree according to the w-uniform distribution of G. The running time of drawing H is dominated by the $O(s)$ calls made to the effective resistance oracle with error $\epsilon = n^{-1/4}$. Invoking Lemma 2.3.4 gives that this cost is bounded by

$$O\left(n\epsilon^{-4} + s\epsilon^{-2}\right) = O\left(n^2\delta^{-2}\right).$$

Furthermore, because Lemma 2.3.4 holds w.h.p. we can absorb the probability of

failure into our total variation distance bound (where we implicitly assume that δ is at most polynomially small).

We then use the $\tilde{O}(m^{1/3}n^{5/3})$ time algorithm in [88] with $m = O(n^{1.5}\delta^{-2})$ to draw a tree from H. This then achieves our desired running time and total variation distance bound.

□

2.6 Implicit Sparsification of the Schur Complement

Note that the determinant sparsification routine in Theorem 2.5.1 only requires an oracle that samples edges by an approximate distribution to resistance, as well as access to approximate leverage scores on the graph. This suggests that a variety of naturally dense objects, such as random walk matrices [47, 111] and Schur complements [41, 88] can also be sparsified in ways that preserve the determinant (of the minor with one vertex removed) or the spanning tree distributions. The latter objects, Schur complements, have already been shown to lead to speedups in random spanning tree generation algorithms recently [88].

Furthermore the fact that Schur complements preserve effective resistances exactly (2.3.6) means that we can directly invoke the effective resistances data structure as constructed in Lemma 2.3.4 to produce effective resistance estimates on any of its Schur complements. As a result, the main focus of this section is an efficient way of producing samples from a distribution that approximates drawing a multi-edge from the Schur complement with probabilities proportional to its leverage score. Here we follow the template introduced in [41] of only eliminating $(1+\alpha)$-diagonally-dominant subsets of vertices, as it in turn allows the use of walk sampling based implicit sparsification similar to those in [47, 111].

$(1+\alpha)$-diagonally-dominant subsets have been used in Schur complement based linear system solvers to facilitate the convergence of iterative methods in the $\boldsymbol{L}_{[V_2,V_2]}$ block [41]. Formally, the condition that we require is:

Definition 2.6.1. In a weighted graph $G = (V, E, w)$, a subset of vertices $V_2 \subseteq V$ is

$(1+\alpha)$-diagonally-dominant, or $(1+\alpha)$-DD if for every $u \in V_2$ with weighted degree $\boldsymbol{d}_u$ we have:

$$\sum_{v \sim u, v \notin V_2} w_{uv} \geq \frac{1}{1+\alpha} \boldsymbol{d}_u = \frac{1}{1+\alpha} \sum_{v \sim u} w_{uv}.$$

It was shown in [41] that large sets of such vertices can be found by trimming a uniformly random sample.

Lemma 2.6.2. *(Lemma 3.5. of [41] instantiated on graphs)*
There is a routine ALMOSTINDEPENDENT(G, α) *that for a graph G with n vertices, and a parameter $\alpha \geq 0$, returns in $O(m)$ expected time a subset V_2 with $|V_2| \geq n/(8(1+\alpha))$ such that $\boldsymbol{L}_{G,[V_2,V_2]}$ is $(1+\alpha)$-DD.*

Given such a subset V_2, we then proceed to sample edges in $\text{Sc}(G, V_1)$ via the following simple random walk sampling algorithm:

1. Pick a random edge in G.
2. Extend both of its endpoints in random walks until they first reach somewhere in V_1.

Incorporating this scheme into the determinant preserving sparsification schemes then leads these guarantees:

Theorem 2.6.3. *Conditioned on Lemma 2.3.4 holding, there is a procedure* SCHURSPARSE *that takes a graph G, and an 1.1-DD subset of vertices V_2, returns a graph H^{V_1} in $\tilde{O}(n^2\delta^{-1})$ expected time such that the distribution over H^{V_1} satisfies:*

$$\mathcal{T}_{\text{Sc}(G,V_1)} \exp(-\delta) \leq \mathbb{E}\left[H^{V_1}\right] \mathcal{T}_{H^{V_1}} \leq \mathcal{T}_{\text{Sc}(G,V_1)} \exp(\delta),$$

and

$$\frac{\mathbb{E}\left[H^{V_1}\right] \mathcal{T}_{H^{V_1}}^2}{\mathbb{E}\left[H^{V_1}\right] \mathcal{T}_{H^{V_1}}{}^2} \leq \exp(\delta).$$

Furthermore, the number of edges of H^{V_1} can be set to anywhere between $O(n^{1.5}\delta^{-1})$ and $O(n^2\delta^{-1})$ without affecting the final bound.

We let this subset of vertices produced to be V_2, and let its complement be V_1. Our key idea is to view $\text{SC}(G, V_1)$ as a multi-graph where each multi-edge corresponds to a walk in G that starts and ends in V_1, but has all intermediate vertices in V_2. Specifically a length k walk

$$u_0, u_1, \ldots u_k,$$

with $u_0, u_k \in V_1$ and $u_i \in V_2$ for all $0 < i < k$, corresponds to a multi-edge between u_0 and u_k in $\text{SC}(G, V_1)$ with weight given by

$$w^{\text{SC}(G,V_1)}_{u_0,u_1,\ldots u_k} \stackrel{\text{def}}{=} \frac{\prod_{0 \leq i < k} w^G_{u_i u_{i+1}}}{\prod_{0<i<k} \boldsymbol{d}^G_{u_i}}. \tag{2.2}$$

We check formally that this multi-graph defined on V_1 is exactly the same as $\text{SC}(G, V_1)$ via the Taylor expansion of $\boldsymbol{L}^{-1}_{[V_2,V_2]}$ based Jacobi iteration.

Lemma 2.6.4. *Given a graph G and a partition of its vertices into V_1 and V_2, the graph G^{V_1} formed by all the multi-edges corresponding to walks starting and ending at V_1, but stays entirely within V_2 with weights given by Equation 2.2 is exactly* $\text{SC}(G, V_1)$.

Proof. Consider the Schur complement:

$$\text{SC}(G, V_1) = \boldsymbol{L}_{[V_1,V_1]} - \boldsymbol{L}_{[V_2,V_1]} \boldsymbol{L}^{\dagger}_{[V_2,V_2]} \boldsymbol{L}_{[V_1,V_2]}.$$

If there are no edges leaving V_2, then the result holds trivially. Otherwise, $\boldsymbol{L}_{[V_2,V_2]}$ is a strictly diagonally dominant matrix, and is therefore full rank. We can write it as

$$\boldsymbol{L}_{[V_2,V_2]} = \boldsymbol{D} - \mathbf{A}$$

where $\boldsymbol{D}$ is the diagonal of $\boldsymbol{L}_{[V_2,V_2]}$ and $\mathbf{A}$ is the negation of the off-diagonal entries, and

then expand $\boldsymbol{L}_{[V_2,V_2]}^{-1}$ via the Jacobi series:

$$\begin{aligned}\boldsymbol{L}_{[V_2,V_2]}^{-1} = (\boldsymbol{D} - \mathbf{A})^{-1} &= \boldsymbol{D}^{-1/2}\left(\boldsymbol{I} - \boldsymbol{D}^{-1/2}\mathbf{A}\boldsymbol{D}^{-1/2}\right)^{-1}\boldsymbol{D}^{-1/2} \\ &= \boldsymbol{D}^{-1/2}\left[\sum_{k=0}^{\infty}\left(\boldsymbol{D}^{-1/2}\mathbf{A}\boldsymbol{D}^{-1/2}\right)^{k}\right]\boldsymbol{D}^{-1/2} = \sum_{k=0}^{\infty}\left(\boldsymbol{D}^{-1}\mathbf{A}\right)^{k}\boldsymbol{D}^{-1}. \quad (2.3)\end{aligned}$$

Note that this series converges because the strict diagonal dominance of $\boldsymbol{L}_{[V_2,V_2]}$ implies $(AD^{-1})^k$ tends to zero as $k \to \infty$. Substituting this in place of $\boldsymbol{L}_{[V_2,V_2]}^{-1}$ gives:

$$\textsc{Sc}\left(G, V_1\right) = \boldsymbol{L}_{[V_1,V_1]} - \sum_{k=0}^{\infty}\boldsymbol{L}_{[V_1,V_2]}\left(\boldsymbol{D}^{-1}\mathbf{A}\right)^{k}\boldsymbol{D}^{-1}\boldsymbol{L}_{[V_2,V_1]}.$$

As all the off-diagonal entries in $\boldsymbol{L}$ are non-positive, we can replace $\boldsymbol{L}_{[V_1,V_2]}$ with $-\boldsymbol{L}_{[V_1,V_2]}$ to make all the terms in the trailing summation positive. As these are the only ways to form new off-diagonal entries, the identity based on matrix multiplication of

$$\left[\left(-\boldsymbol{L}_{[V_1,V_2]}\right)\left(\boldsymbol{D}^{-1}\mathbf{A}\right)^{k}\boldsymbol{D}^{-1}\left(-\boldsymbol{L}_{[V_2,V_1]}\right)\right]_{u_0,u_k} = \sum_{u_1\ldots u_{k-1}}\frac{\prod_{0\leq i<k} w_{u_i u_{i+1}}^{G}}{\prod_{0<i<k}\boldsymbol{d}_{u_i}^{G}}$$

gives the required identity. □

This characterization of $\textsc{Sc}\left(G, V_1\right)$, coupled with the $(1+\alpha)$-diagonal-dominance of V_2, allows us to sample the multi-edges in $\textsc{Sc}\left(G, V_1\right)$ in the same way as the (short) random walk sparsification algorithms from [47, 111].

Lemma 2.6.5. *Given any graph $G = (V, E, w)$, an $(1+\alpha)$-DD subset V_2, and access to 2-approximations of statistical leverage scores on G, $\widetilde{\boldsymbol{\tau}}^{G}$,* SAMPLEEDGESCHUR *returns edges in G according to the distribution $\boldsymbol{p}_e$ in $O(\alpha)$ expected time per sample. Furthermore, the distribution that it samples edges in $\textsc{Sc}\left(G, V_1\right)$ from, $\boldsymbol{p}$, satisfies*

$$O\left(1\right)\cdot\boldsymbol{p}_{u_0,\ldots u_k} \geq \frac{\overline{\boldsymbol{\tau}}_{u_0,\ldots u_k}^{\textsc{Sc}(G,V_1)}}{n-1}.$$

Algorithm 3: SAMPLEEDGESCHUR$(G = (V, E, w), V_1)$: samples an edge from $\mathrm{SC}(G, V_1)$

Input: Graph G, vertices V_1 to complement onto, and (implicit) access to a 2-approximation of the leverage scores of G, $\widetilde{\boldsymbol{\tau}}^G$.

Output: A multi-edge e in $\mathrm{SC}(G, V_1)$ corresponding to a walk $u_0, u_1, \ldots u_k$, and the probability of it being picked in this distribution $\boldsymbol{p}_{u_0,u_1,\ldots u_k}$

1 Sample an edge e from G randomly with probability drawn from $\widetilde{\boldsymbol{\tau}}^G_e$;

2 Perform two independent random walks from the endpoints of e until they both reach some vertex in V_1, let the walk be $u_0 \ldots u_k$;

3 Output edge u_0u_k (corresponding to the path $u_0, u_1, \ldots u_k$) with

$$w_{u_0\ldots u_k} \leftarrow \frac{\prod_{0\le i<k} w^G_{u_iu_{i+1}}}{\prod_{0<i<k} \boldsymbol{d}^G_{u_i}}, \qquad \text{(same as Equation 2.2)}$$

$$\boldsymbol{p}_{u_0\ldots u_k} \leftarrow \frac{1}{\sum_{e'} \widetilde{\boldsymbol{\tau}}^G_{e'}} \sum_{0\le i<k} \widetilde{\boldsymbol{\tau}}^G_{u_iu_{i+1}} \cdot \left(\prod_{0\le j<i} \frac{w^G_{u_ju_{j+1}}}{\boldsymbol{d}_{u_{j+1}}} \cdot \prod_{i+1\le j<k} \frac{w^G_{u_ju_{j+1}}}{\boldsymbol{d}_{u_j}} \right).$$

for every edge in $\mathrm{SC}(G, V_1)$ *corresponding to the walk* $u_0, \ldots u_k$.

The guarantees of this procedure are analogous to the random walk sampling sparsification scheme from [47, 111], with the main difference being the terminating condition for the walks leads to the removal of an overhead related to the number of steps in the walk. The modification of the initial step to picking the initial edge from G by resistance is necessary to get ρ to a constant, as the about $n^{1.5}$ samples limits the amount of overhead that we can have per sample.

Proof. We first verify that $\boldsymbol{p}$ is indeed a probability on the multi-edges of $\mathrm{SC}(G, V_1)$, partitioned by the walks that they correspond to in G, or formally

$$\sum_{\substack{u_0,u_1,\ldots u_k:\\ u_0,u_k\in V_1,\\ u_i\in V_2\ \forall 1\le i<k}} \boldsymbol{p}_{u_0,u_1\ldots u_k} = 1.$$

To obtain this equality, note that for any random walk starting at vertex i, the total probabilities of walks starting at i and ending in V_1 is upper bounded by 1. Algebraically this

becomes:

$$\sum_{u_1,u_2,\ldots u_k} \prod_{0\leq i<k} \frac{w_{u_iu_{i+1}}}{\boldsymbol{d}_{u_i}} = 1,$$

so applying this to both terms of each edge e gives that the total probability mass over any starting edge is $\frac{\widetilde{\boldsymbol{\tau}}_e^G}{\sum_{e'} \widetilde{\boldsymbol{\tau}}_{e'}^G}$, and in turn the total.

For the running time, since V_2 is $(1+\alpha)$-almost independent, each step of the walk takes expected time $O(\alpha)$. Also, the value of $\boldsymbol{p}_{u_0,u_1,\ldots u_k}$ can be computed in $O(k)$ time by computing prefix/suffix products of the transition probabilities along the path (instead of evaluating each summand in $O(k)$ time for a total of $O(k^2)$).

Finally, we need to bound the approximation of $\boldsymbol{p}$ compared to the true leverage scores $\overline{\boldsymbol{\tau}}$. As $\widetilde{\boldsymbol{\tau}}_e^G$ is a 2-approximation of the true leverage scores, $\sum_e \widetilde{\boldsymbol{\tau}}_e^G$ is within a constant factor of n. So it suffices to show

$$O(1) \cdot \sum_{0\leq i<k} \widetilde{\boldsymbol{\tau}}_{u_iu_{i+1}}^G \left(\prod_{0\leq j<i} \frac{w_{u_ju_{j+1}}^G}{\boldsymbol{d}_{u_{j+1}}} \cdot \prod_{i+1\leq j<k} \frac{w_{u_ju_{j+1}}^G}{\boldsymbol{d}_{u_j}} \right) \geq \mathcal{R}_{eff}^{\mathrm{Sc}(G,V_1)}(u_0,u_k) \cdot w_{u_0,u_1,\ldots u_k}.$$

Here we invoke the equivalence of effective resistances in G and $\mathrm{Sc}(G,V_1)$ given by Fact 2.3.6 in the reverse direction. Then by Rayleigh's monotonicity principle, we have

$$\mathcal{R}_{eff}^{\mathrm{Sc}(G,V_1)}(u_0,u_k) = \mathcal{R}_{eff}^{G}(u_0,u_k) \leq \sum_{0\leq i<k} \frac{2\widetilde{\boldsymbol{\tau}}_{u_iu_{i+1}}^G}{w_{u_iu_{i+1}}},$$

which when substituted into the expression for $w_{u_0,u_1,\ldots u_k}$ from Equation 2.2 gives

$$\left(\sum_{0\leq i<k} \frac{2\widetilde{\boldsymbol{\tau}}_{u_iu_{i+1}}^G}{w_{u_iu_{i+1}}} \right) w_{u_0,u_1,\ldots u_k} = \sum_{0\leq i<k} 2\widetilde{\boldsymbol{\tau}}_{u_iu_{i+1}}^G \left(\prod_{0\leq j<i} \frac{w_{u_ju_{j+1}}^G}{\boldsymbol{d}_{u_{j+1}}} \cdot \prod_{i+1\leq j<k} \frac{w_{u_ju_{j+1}}^G}{\boldsymbol{d}_{u_j}} \right).$$

□

This sampling procedure can be immediately combined with Theorem 2.5.1 to give algorithms for generating approximate Schur complements. Pseudocode of this routine is in Algorithm 4.

Algorithm 4: SCHURSPARSE(G, V_1, δ)

Input: Graph G, 1.1-DD subset of vertices V_2 and error parameter δ
Output: Sparse Schur complement of SC (G, V_1)

Set $\epsilon \leftarrow 0.1$;
Set $s \leftarrow n^2\delta^{-1}$;
Build leverage score data structure on G with errors 0.1 (via Lemma 2.3.4);
Let $H^{V_1} \leftarrow$
DETSPARSIFY(SC (G, V_1), s, SAMPLEEDGESCHUR(G, V_1), LEVERAGEAPPROX$_G$, ϵ);
Output H^{V_1};

Proof. (Of Theorem 2.6.3) Note that the choices of ϵ and s must ensure that

$$\frac{n^2\epsilon^2}{s} = \delta$$
$$\frac{n^3}{s^2} \leq \delta$$

This is then equivalent to $s \geq n^{1.5}\delta^{-1}$ and $\frac{s}{\epsilon^2} = n^2\delta^{-1}$. This further implies that $\epsilon \geq n^{1/4}$. Our ϵ and s in SCHURSPARSE meet these conditions (and the ones specifically chosen in the algorithm will also be necessary for one of our applications). The guarantees then follow from putting the quality of the sampler from Lemma 2.6.5 into the requirements of the determinant preserving sampling procedure from Theorem 2.5.1. Additionally, Lemma 2.6.5 only requires access to 2-approximate leverage scores, which can be computed by Lemma 2.3.4 in $\tilde{O}(m)$ time. Furthermore, Lemma 2.6.5 gives that our ρ value is constant, and our assumption in Theorem 2.6.3 that we are given an 1.1-DD subset V_2 implies that our expected $O(s \cdot \rho)$ calls to SAMPLEEDGESCHUR will require $O(1)$ time. The only other overheads are the computation and invocations of the various copies of approximate resistance data structures. Since $m \leq n^2$ and $\epsilon \geq n^{1/4}$, Lemma 2.3.4 gives that this cost is bounded by $\tilde{O}(m + n^2 + \frac{s}{\epsilon^2}) = \tilde{O}(n^2\delta^{-1})$. □

2.7 Approximate Determinant of SDDM Matrices

In this section, we provide an algorithm for computing an approximate determinant of SDDM matrices, which are minors of graph Laplacians formed by removing one row/column.

Theorem 2.2.1 allows us to sparsify a dense graph while still approximately preserving the determinant of the graph minor. If there were some existing algorithm for computing the determinant that had good dependence on sparsity, we could achieve an improved runtime for determinant computation by simply invoking such an algorithm on a minor of the sparsified graph.[6] Unfortunately, current determinant computation algorithms (that achieve high-accuracy) are only dependent on n, so simply reducing the edge count does not directly improve the runtime for determinant computation. Instead the algorithm we give will utilize Fact 2.3.5

$$\det_+(\boldsymbol{L}) = \det\left(\boldsymbol{L}_{[V_2,V_2]}\right) \cdot \det_+(\textsc{Sc}\left(\boldsymbol{L}, V_1\right)).$$

(where we recall that $\det_+$ is the determinant of the matrix minor) to recursively split the matrix. Specifically, we partition the vertex set based upon the routine ALMOSTINDEPENDENT from Lemma 2.6.2, then compute Schur complements according to SCHURSPARSE in Theorem 2.6.3. Our algorithm will take as input a Laplacian matrix. However, this recursion naturally produces two matrices, the second of which is a Laplacian and the first of which is a submatrix of a Laplacian. Therefore, we need to convert $\boldsymbol{L}_{[V_2,V_2]}$ into a Laplacian. We do this by adding one vertex with appropriate edge weights such that each row and column sums to 0. Pseudocode of this routine is in Algorithm 5, and we call it with the parameters $\boldsymbol{L}^{V_2} \leftarrow \textsc{AddRowColumn}(\boldsymbol{L}_{[V_2,V_2]})$.

The procedure ADDROWCOLUMN outputs a Laplacian $\boldsymbol{L}^{V_2}$ such that $\boldsymbol{L}_{[V_2,V_2]}$ can be obtained if one removes this added row/column. This immediately gives $\det_+(\boldsymbol{L}^{V_2}) =$

[6] To get with high probability one could use standard boosting tricks involving taking the median of several estimates of the determinant obtained in this fashion.

Algorithm 5: ADDROWCOLUMN($\boldsymbol{M}$) : complete $\boldsymbol{M}$ into a graph Laplacian by adding one more row/column

Input: SDDM Matrix $\boldsymbol{M}$
Output: Laplacian matrix $\boldsymbol{L}$ with one extra row / column than $\boldsymbol{M}$

Let n be the dimension of $\boldsymbol{M}$;
for $i = 1$ *to* n **do**
    Sum non-zero entries of row i, call $\boldsymbol{s}_i$;
    Set $\boldsymbol{L}(n+1, i), \boldsymbol{L}(i, n+1) \leftarrow -\boldsymbol{s}_i$;
Let $\boldsymbol{L}(n+1, n+1) \leftarrow \sum_{i=1}^{n} \boldsymbol{s}_i$;
Output $\boldsymbol{L}$;

$\det(\boldsymbol{L}_{[V_2,V_2]})$ by definition, and we can now give our determinant computation algorithm of the minor of a graph Laplacian.

Algorithm 6: DETAPPROX($\boldsymbol{L}, \delta, \overline{n}$) : Compute $\det_+(\boldsymbol{L})$ with error parameter δ

Input: Laplacian matrix $\boldsymbol{L}$, top level error threshold δ, and top level graph size $\overline{n}$
Output: Approximate $\det_+(\boldsymbol{L})$

if *this is the top-level invocation of this function in the recursion tree* **then**
    $\delta' \leftarrow \Theta(\delta^2 / \log^3 n)$
else
    $\delta' \leftarrow \delta$
if $\boldsymbol{L}$ *is* 2×2 **then**
    return *the weight on the (unique) edge in the graph*
$V_2 \leftarrow$ ALMOSTINDEPENDENT($\boldsymbol{L}, \frac{1}{10}$) {Via Lemma 2.6.2}
$V_1 \leftarrow V \setminus V_2$;
$\boldsymbol{L}^{V_1} \leftarrow$ SCHURSPARSE($\boldsymbol{L}, V_1, \delta'$); {$|V_1| / \overline{n}$ is the value of β in Lemma 2.7.1.}
$\boldsymbol{L}^{V_2} \leftarrow$ ADDROWCOLUMN($\boldsymbol{L}_{[V_2,V_2]}$);
Output DETAPPROX($\boldsymbol{L}^{V_1}, \delta' |V_1| / \overline{n}, \overline{n}$) $\cdot$ DETAPPROX($\boldsymbol{L}^{V_2}, \delta' |V_2| / \overline{n}, \overline{n}$);

Our analysis of this recursive routine consists of bounding the distortions incurred at each level of the recursion tree. This in turn uses the fact that the number of vertices across all calls within a level and the total "amount" of δ across all calls within a level both remain unchanged from one level to the next. This can be summarized by the following Lemma which bounds the error accumulated within one level of recursion in our algorithm.

Lemma 2.7.1. *Suppose we are given some small $\delta \geq 0$ and non-negative $\beta_1, ..., \beta_k$ such that $\sum_{i=1}^{k} \beta_i = O(1)$, along with Laplacian matrices $\boldsymbol{L}(1), \ldots, \boldsymbol{L}(k)$ and each having a*

corresponding vertex partition $V_1(i), V_2(i)$, where

$$\boldsymbol{L}(i) = \begin{bmatrix} \boldsymbol{L}(i)_{[V_1(i),V_1(i)]} & \boldsymbol{L}(i)_{[V_1(i),V_2(i)]} \\ \boldsymbol{L}(i)_{[V_2(i),V_1(i)]} & \boldsymbol{L}(i)_{[V_2(i),V_2(i)]} \end{bmatrix}.$$

Let $\boldsymbol{L}^{V_1(i)}$ denote the result of running SCHURSPARSE *to remove the $V_2(i)$ block in each of these matrices:*[7]

$$\boldsymbol{L}^{V_1(i)} \stackrel{\text{def}}{=} \textsc{SchurSparse}\left(\boldsymbol{L}(i), V_1(i), \beta_i \delta\right).$$

Then conditioning upon a with high probability event[8] *in each of these calls to* SCHURSPARSE*, for any p we have with probability at least $1 - p$:*

$$\prod_{i=1}^{k} \det_+\left(\boldsymbol{L}(i)\right) = \left(1 \pm O\left(\sqrt{\delta/p}\right)\right) \prod_{i=1}^{k} \det\left(\boldsymbol{L}_{[V_2(i),V_2(i)]}(i)\right) \cdot \det_+\left(\boldsymbol{L}^{V_1(i)}\right).$$

Here the β_i corresponds to the $|V_1|/n$ and $|V_2|/n$ values that δ is multiplied against in each call parameter to SCHURSPARSE. An example of the main steps in this determinant approximation algorithm, as well as the graphs corresponding to applying Lemma 2.7.1 to one of the layers is in Figure 2.1.

Applying Lemma 2.7.1 to all the layers of the recursion tree gives the overall guarantees.

Proof of Theorem 2.2.2.

Running Time: Let the number of vertices and edges in the current graph corresponding to $\boldsymbol{L}$ be n and m respectively. Calling ALMOSTINDEPENDENT takes expected time $O(m)$ and guarantees

$$\frac{n}{16} \leq |V_2| \leq \frac{n}{8},$$

[7]This Lemma only applies when the matrices are fixed with respect to the randomness used in the invocations of SCHURSPARSE mentioned in the Lemma. In other words, it only applies when the result of running SCHURSPARSE on each of these $\boldsymbol{L}(i)$ matrices is independent of the result of running it on the other matrices. This is why the Lemma only immediately bounds error within a level of the recursion—where this independence holds—rather than for the entire algorithm.

[8]namely, the event that all the leverage score estimation calls to Lemma 2.3.4 from SCHURSPARSE succeed

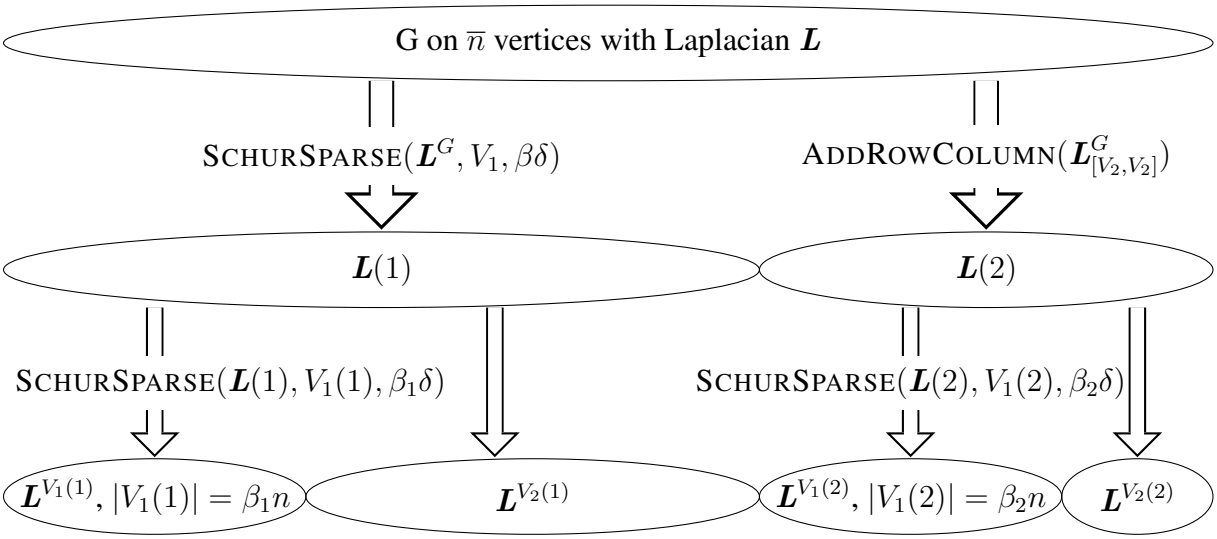

Figure 2.1: Two layers of the call Structure of the determinant approximation algorithm DETAPPROX (algorithm 6), with the transition from the first to the second layer labeled as in Lemma 2.7.1.

which means the total recursion terminates in $O(\log n)$ steps.

For the running time, note that as there are at most $O(n)$ recursive calls, the total number of vertices per level of the recursion is $O(n)$. The running time on each level are also dominated by the calls to SCHURSPARSE, which comes out to

$$\tilde{O}\left(|V_1(i)|^2 \frac{n}{\delta' |V_1(i)|}\right) = \tilde{O}\left(|V_1(i)| n\delta^{-2}\right),$$

and once again sums to $\tilde{O}(n^2\delta^{-2})$. We note that this running time can also be obtained from more standard analyses of recursive algorithms, specifically applying guess-and-check to a running time recurrence of the form of:

$$T(n, \delta) = T(\theta n, \theta\delta) + T((1-\theta)n + 1, (1-\theta)\delta) + \tilde{O}(n^2\delta^{-1}).$$

Correctness. As shown in the running time analysis, our recursion tree has depth at most $O(\log n)$, and there are at most $O(n)$ total vertices at any given level. We associate each level of the recursion in our algorithm with the list of matrices which are given as input

to the calls making up that level of recursion. For any level in our recursion, consider the product of $\det_+$ applied to each of these matrices. We refer to this quantity for level j as q_j. Notice that q_0 is the determinant we wish to compute and $q_{\#\text{ levels}-1}$ is what our algorithm actually outputs. As such, it suffices to prove that for any j, $q_j = (1 \pm \frac{\delta}{\#\text{ levels}})q_{j-1}$ with probability of failure at most $\frac{1}{10 \cdot \#\text{ levels}}$. However, by the fact that we set $\delta' = \Theta(\delta^2/\log^3 n)$ in the top level of recursion with sufficiently small constants, this immediately follows from Lemma 2.7.1.

A minor technical issue is that Lemma 2.7.1 only gives guarantees conditioned on a WHP event. However, we only need to invoke this Lemma a logarithmic number of times, so we can absorb this polynomially small failure probability into the our total failure probability without issue.

Standard boosting techniques—such as running $O(\log n)$ independent instances and taking the medians of the estimates— give our desired with high probability statement. □

It remains to bound the variances per level of the recursion.

Proof. (Of Lemma 2.7.1) As a result of Fact 2.3.5

$$\prod_{i=1}^{k} \det_+ \left(\boldsymbol{L}(i)\right) = \prod_{i=1}^{k} \det\left(\boldsymbol{L}(i)_{[V_2(i),V_2(i)]}\right) \det_+ \left(\textsc{Sc}\left(\boldsymbol{L}(i), V_1(i)\right)\right).$$

Consequently, it suffices to show that with probability at least $1 - p$

$$\prod_{i=1}^{k} \det_+ \left(\textsc{Sc}\left(\boldsymbol{L}(i), V_1(i)\right)\right) = \left(1 \pm O\left(\sqrt{\delta/p}\right)\right) \prod_{i=1}^{k} \det_+ \left(\boldsymbol{L}^{V_1(i)}\right).$$

Recall that $\boldsymbol{L}^{V_1(i)}$ denotes the random variable that is the approximate Schur complement generated through the call to SCHURSPARSE$(\boldsymbol{L}(i), V_1(i), \beta_i\delta)$.

Using the fact that our calls to SCHURSPARSE are independent along with the assump-

tion of $\sum_{i=1}^{k} \beta_i = O(1)$, we can apply the guarantees of Theorem 2.6.3 to obtain

$$\mathbb{E}\left[\boldsymbol{L}^{V_1(1)} \dots \boldsymbol{L}^{V_1(k)}\right] \prod_{i=1}^{k} \det{}_+\left(\boldsymbol{L}^{V_1(1)}\right) = \prod_{i=1}^{k} \mathbb{E}\left[\boldsymbol{L}^{V_1(i)}\right] \det{}_+ \boldsymbol{L}^{V_1(i)}$$
$$= (1 \pm O(\delta)) \prod_{i=1}^{k} \det{}_+ \left(\textsc{Sc}\left(\boldsymbol{L}(i), V_1(i)\right)\right),$$

and

$$\frac{\mathbb{E}\left[\boldsymbol{L}^{V_1(1)} \dots \boldsymbol{L}^{V_1(k)}\right] \prod_{i=1}^{k} \det{}_+\left(\boldsymbol{L}^{V_1(i)}\right)^2}{\mathbb{E}\left[\boldsymbol{L}^{V_1(1)} \dots \boldsymbol{L}^{V_1(k)}\right] \prod_{i=1}^{k} \det{}_+\left(\boldsymbol{L}^{V_1(i)}\right)^2} = \prod_{i=1}^{k} \frac{\mathbb{E}\left[\boldsymbol{L}^{V_1(i)}\right] \det{}_+\left(\boldsymbol{L}^{V_1(i)}\right)^2}{\mathbb{E}\left[\boldsymbol{L}^{V_1(i)}\right] \det{}_+\left(\boldsymbol{L}^{V_1(i)}\right)^2}$$
$$\leq \prod_{i=1}^{k} \exp\left(O\left(\beta_i \delta\right)\right) \leq \exp\left(O\left(\delta\right)\right).$$

By assumption δ is small, so we can approximate $\exp\left(O(\delta)\right)$ with $1 + O(\delta)$, which with bound above gives

$$\textbf{Var}\left[\boldsymbol{L}^{V_1(1)} \dots \boldsymbol{L}^{V_1(k)}\right] \prod_{i=1}^{k} \det{}_+\left(\boldsymbol{L}^{V_1(i)}\right) \leq O\left(\delta\right) \mathbb{E}\left[\boldsymbol{L}^{V_1(1)} \dots \boldsymbol{L}^{V_1(k)}\right] \prod_{i=1}^{k} \det{}_+\left(\boldsymbol{L}^{V_1(i)}\right)^2,$$

Then applying the approximation on $\mathbb{E}\left[\right] \prod_{i=1}^{k} \det{}_+ \left(\textsc{SchurSparse}(\boldsymbol{L}(i), V_1(i), \beta_i \delta)\right)$ gives

$$\textbf{Var}\left[\boldsymbol{L}^{V_1(1)} \dots \boldsymbol{L}^{V_1(k)}\right] \prod_{i=1}^{k} \det{}_+\left(\boldsymbol{L}^{V_1(i)}\right) \leq O\left(\delta\right) \left(\prod_{i=1}^{k} \det{}_+ \left(\textsc{Sc}\left(\boldsymbol{L}(i), V_1(i)\right)\right)\right)^2.$$

At which point we can apply Chebyshev's inequality to obtain our desired result.

□

2.8 Random Spanning Tree Sampling

In this section we will give an algorithm for generating a random spanning tree from a weighted graph, that uses SCHURSPARSE as a subroutine, and ultimately prove Theo-

rem 2.2.3.

In order to do so, we will first give an $O(n^\omega)$ time recursive algorithm using Schur complement that exactly generates a random tree from the w-uniform distribution. The given algorithm is inspired by the one introduced in [108], and its variants utilized in [103, 104, 88]. However, we will (out of necessity for our further extensions) reduce the number of branches in the recursion to two, by giving an efficient algorithmic implementation of a bijective mapping between spanning trees in G and spanning trees in $\textsc{Sc}(G, V_2)$ when V_1, the set of vertices removed, is an independent set. We note that this also yields an alternative algorithm for generating random spanning trees from the w-uniform distribution in $O(n^\omega)$ time.

The runtime of this recursion will then be achieved similar to our determinant algorithm. We reduce δ proportional to the decrease in the number of vertices for every successive recursive call in exactly the same was as the determinant approximation algorithm from Section 2.7. As has been previously stated and which is proven in Section 2.10.1, drawing a random spanning tree from a graph after running our sparsification routine which takes $\widetilde{O}(n^2\delta^{-1})$, will have total variation distance $\sqrt{\delta}$ from the w-uniform distribution.

Similar to our analysis of the determinant algorithm, we cannot directly apply this bound to each tree because the lower levels of the recursion will contribute far too much error when δ is not decreasing at a proportional rate to the total variation distance. Thus we will again need to give better bounds on the variance across each level, allowing stronger bounds on the contribution to total variation distance of the entire level.

This accounting for total variance is more difficult here due to the stronger dependence between the recursive calls. Specifically, the input to the graph on V_2 depends on the set of edges chosen in the first recursive call on V_1, specifically $\textsc{Sc}(G, V_1)$, or a sparsified version of it.

Accounting for this dependency will require proving additional concentration bounds shown in Section 2.9, which we specifically achieve by sampling $s = O(n^2\delta^{-1})$ edges

in each call to SCHURSPARSE. While this might seem contradictory to the notion of "sampling", we instead consider this to be sampling from the graph in which all the edges generated from the Schur complement are kept separate and could be far more than n^2 edges.

2.8.1 Exact $O(n^\omega)$ Time Recursive Algorithm

We start by showing an algorithm that samples trees from the exact w-uniform distribution via the computation of Schur complements. Its pseudocode is in Algorithm 7, and it forms the basis of our approximate algorithm: the faster routine in Section 2.8.2 is essentially the same as inserting sparsification steps between recursive calls.

Algorithm 7: EXACTTREE(G) : Take a graph and output a tree randomly from the w-uniform distribution

Input: Graph G
Output: A tree randomly generated from the w-uniform distribution of G

If there is only one edge e in G, return G ;
Partition V evenly into V_1 and V_2;
$T_1 = \text{EXACTTREE}(\text{SC}(G, V_1))$;
for *each* $e \in T_1$ **do**
    with probability $\frac{w_e(G)}{w_e(\text{SC}(G,V_1))}$, $G \leftarrow G/e$, $T \leftarrow T \cup e$;
Delete the remaining edges, i.e., $G \leftarrow G \setminus E(V_1)$;
$T_2 = \text{EXACTTREE}(\text{SC}(G, V_2))$;
$T \leftarrow T \cup \text{PROLONGATETREE}(G, V_1 \sqcup V_2, T_2)$;
Output T;

The procedure PROLONGATETREE is invoked when $V_1 = V \setminus V_2$ maps a tree T_2 from the Schur complement SC (G, V_2) to a tree back in G. It crucially uses the property that V_1 is an independent set, and its pseudocode is given in Algorithm 8.

Lemma 2.8.1. *The procedure* EXACTTREE(G) *will generate a random tree of* G *from the* w*-uniform distribution in* $O(n^\omega)$ *time.*

The algorithm we give is similar to the divide and conquer approaches of [108, 103, 104, 88]. The two main facts used by these approaches can be summarized as follows:

Algorithm 8: PROLONGATETREE$(G, V_1 \sqcup V_2, T_2)$: prolongating a tree on $\mathrm{SC}(G, V_2)$ to a tree on G.

Input: A graph G, a splitting of vertices $V_1 \sqcup V_2$ such that V_1 is an independent set, tree T_2 of $\mathrm{SC}(G, V_2)$.

Output: A tree in G

$T \leftarrow \emptyset$;
for *each* $e = xy \in T_2$ **do**
    Create distribution λ_e, set $\lambda_e(\emptyset) = w_e(G)$;
    for *each* $v \in V_1$ *such that* $(v, x), (v, y) \in E(G)$ **do**
        Set $\lambda_e(v) = w_{(v,x)}(G) w_{(v,y)}(G) \boldsymbol{d}_v(G)^{-1}$;
    Randomly assign $f(e)$ to $\{\emptyset \cup V_1\}$ with probability proportional to λ;
for *each* $v \in V_1$ **do**
    for *each* $e = (x, y) \in T_2$ *such that* $(v, x), (v, y) \in E(G)$ **do**
        if $f(e) \neq v$ **then**
            Contract x and y ;
    for *each contracted vertex* X *in the neighborhood of* v **do**
        Connect X to v with edge $(v, u) \in G$ with probability proportional to $w_G((v, u))$;
        $T \leftarrow T \cup (v, u)$;
Output T;

1. Schur complements preserves the leverage score of original edges, and

2. The operation of taking Schur complements, and the operation of deleting or contracting an edge are associative.

We too will make use of these two facts. But unlike all previous approaches, at every stage we need to recurse on only two sub-problems. All previous approaches have a branching factor of at least four.

We can do this by exploiting the structure of the Schur complement when one eliminates an independent set of vertices. We formalize this in Lemma 2.8.5.

Before we can prove the lemma, we need to state an important property of Schur complements that follows from Fact 2.3.8. Recall the notation from Section 2.3 that for a weighted graph $G = (V, E, w)$, $\boldsymbol{Pr}_T^G(\cdot)$ denotes the probability of $\cdot$ over trees T picked from the w-uniform distribution on spanning trees of G.

Lemma 2.8.2. *Let G be a graph with a partition of vertices $V = V_1 \sqcup V_2$. Then for any set of edges F contained in $G[V_1]$, the induced subgraph on V_1, we have:*

$$\boldsymbol{Pr}_T^G\left(T \cap E\left(G\left[V_1\right]\right) = F\right) = \boldsymbol{Pr}_T^{\mathrm{Sc}(G,V_1)}\left(T \cap E\left(G\left[V_1\right]\right) = F\right),$$

where the edges in $\mathrm{Sc}\left(G, V_1\right)$ are treated as the sum of $G[V_1]$ and $G_{sc}[V_1]$, the new edges added to the Schur complement.

Proof. If F contains a cycle, then $\boldsymbol{Pr}_T^G\left(T \cap E\left(G\left[V_1\right]\right) = F\right) = 0 = \boldsymbol{Pr}_T^{\mathrm{Sc}(G,V_1)}(T \cap E\left(G\left[V_1\right]\right) = F)$. Therefore, we will assume F does not contain any cycle, and we will prove by induction on the size of F. If $|F| > |V_1| - 1$, then F will have to contain a cycle. When $|F| = |V_1| - 1$, then F will have to be the edge set of a tree in $\mathrm{Sc}(G, V_1)$. Then by Fact 2.3.8, the corollary holds. Now suppose that the corollary holds for all F with

$|F| = |V_1| - 1 - k$. Now consider some F with $|F| = |V_1| - 1 - (k+1)$. We know

$$\boldsymbol{Pr}_T^G\left(F \subseteq T\right) = \boldsymbol{Pr}_T^G\left(F = \left(T \cap E\left(G\left[V_1\right]\right)\right)\right) + \sum_{F' \supset F} \boldsymbol{Pr}_T^G\left(F' = \left(T \cap E\left(G\left[V_1\right]\right)\right)\right).$$

Since $|F'| > |F|$, by assumption

$$\sum_{F' \supset F} \boldsymbol{Pr}_T^G\left(F' = \left(T \cap E\left(G\left[V_1\right]\right)\right)\right) = \sum_{F' \supset F} \boldsymbol{Pr}_T^{\mathrm{Sc}(G,V_1)}\left(F' = \left(T \cap E\left(G\left[V_1\right]\right)\right)\right),$$

then by Fact 2.3.8 we have $\boldsymbol{Pr}_T^G(F \subseteq T) = \boldsymbol{Pr}_T^{\mathrm{Sc}(G,V_1)}(F \subseteq T)$, which implies

$$\boldsymbol{Pr}_T^G\left(F = \left(T \cap E\left(G\left[V_1\right]\right)\right)\right) = \boldsymbol{Pr}_T^{\mathrm{Sc}(G,V_1)}\left(F = \left(T \cap E\left(G\left[V_1\right]\right)\right)\right).$$

□

The tracking of edges from various layers of the Schur complement leads to another layer of overhead in recursive algorithms. They can be circumvented by merging the edges, generating a random spanning tree, and the 'unsplit' the edge by random spanning. The following is a direct consequence of the definition of $w(T)$:

Lemma 2.8.3. *Let $\widehat{G}$ be a multi-graph, and G be the simple graph formed by summing the weights of overlapping edges. Then the procedure of:*

1. *Sampling a random spanning tree from G, T.*
2. *For each edge $e \in T$, assign it to an original edge from $\widehat{G}$, $\widehat{e}$ with probability*

$$\frac{w_{\widehat{e}}\left(\widehat{G}\right)}{w_e\left(G\right)}.$$

Produces a w-uniform spanning tree from $\widehat{G}$, the original multi-graph.

This then leads to the following proto-algorithm:

1. Partition the vertices (roughly evenly) into

$$V = V_1 \sqcup V_2.$$

2. Generate a w-uniform tree of $\text{Sc}(G, V_1)$, and create $F_1 = T \cap E(G[V_1])$ by re-sampling edges in $G[V_1]$ using Lemma 2.8.3. By Lemma 2.8.2, this subset is precisely the intersection of a random spanning tree with $G[V_1]$.

3. This means we have 'decided' on all edges in $G[V_1]$. So we can proceed by contracting all the edges of F_1, and deleting all the edges corresponding to $E(G[V_1])\backslash F$. Let the resulting graph be G' and let V_1' be the remaining vertices in V_1 after this contraction.

4. Observe that V_1' is an independent set, and its complement is V_2. We can use another recursive call to generate a w-uniform tree in $\text{Sc}(G', V_2)$. Then we utilize the fact that V_1' is an independent set to lift this to a tree in G' efficiently via Lemma 2.8.5.

Our key idea for reducing the number of recursive calls of the algorithm, that when V_1 (from the partition of vertices $V = V_1 \sqcup V_2$) is an independent set, we can directly lift a tree from $\text{Sc}(G, V_2)$ to a tree in G. This will require viewing $G_{\text{Sc}}[V_2]$ as a sum of cliques, one per vertex of V_1, plus the original edges in $G[V_2]$.

Fact 2.8.4. *Given a graph G and a vertex v, the graph $\text{Sc}(G, V \setminus v)$ is the induced graph $G[V \setminus \{v\}]$ plus a weighted complete graph $K(v)$ on the neighbors of v. This graph $K(v)$ is formed by adding one edge xy for every pair of x and y incident to v with weight*

$$\frac{w_{(v,x)} w_{(v,y)}}{deg_v},$$

where $\boldsymbol{d}_v \stackrel{\text{def}}{=} \sum_x w_{(v,x)}$ is the weighted degree of v in G.

Lemma 2.8.5. *Let G be a graph on n vertices and V_1 an independent set. If T is drawn from the w-uniform distribution of $\text{Sc}(G, V_2)$, then in $O(n^2)$ time* ProlongateTree$(G, V_1 \sqcup$

V_2, T_2) *returns a tree from the* w*-uniform distribution of* G.

Proof. The running time of PROLONGATETREE is $O(n^2)$ as T_2 has $\leq n-1$ edges and $|V_1| \leq n$.

Now we will show the correctness. Let $V_1 = \{v_1, ..., v_k\}$. We will represent $\text{SC}(G, V_2)$ as a multi-graph arising by Schur complementing out the vertices in V_1 one by one and keeping the new edges created in the process separate from each other as a multi-graph. We represent this multi-graph as

$$\text{SC}(G, V_2) = G[V_2] + K(v_1) + ... + K(v_k),$$

where $G[V_2]$ is the induced subgraph on V_2 and $K(v_i)$ is the weighted complete graph on the neighbors of v_i. Then

- By the unsplitting procedure from Lemma 2.8.3, the function f maps T_2 to a tree in the multi-graph $G[V_2] + K(v_1) + ... + K(v_k)$, and
- the rest of the sampling steps maps this tree to one in G.

We will now prove correctness by induction on the size of the independent set V_1. The case of $|V_1| = 0$ follows from $\text{SC}(G, V_2) = G$. If $|V_1| = 1$, i.e, $V_1 = \{v\}$ for some vertex v, then $\text{SC}(G, V_2)$ is $G[V_2] + K(v)$. Given a tree T_2 of $\text{SC}(G, V_2)$, the creation of f will first map T_2 to a tree in the multigraph $G[V_2] + K(v)$ by randomly deciding for each edge $e \in T$ to be in $G(V_1)$ or $K(v)$ depending on it's weight. If we let $T'(V_2) = T' \cap G[V_2]$, then by Lemma 2.8.2,

$$\boldsymbol{Pr}_T^G\left(T \cap E(G[V_2]) = T'(V_2)\right) = \boldsymbol{Pr}_T^{G[V_2]+K(v)}\left(T \cap E(G[V_2]) = T'(V_2)\right).$$

Therefore, we can contract all the edges of $T'(V_2) \cap G[V_2]$ and delete all other edges of $G[V_2]$. This results in a multi-graph star with v at the center. Now, PROLONGATETREE does the following to decide on the remaining edges. For every multi-edge of the star graph

obtained by contracting or deleting edges in $G[V_2]$, we choose exactly one edge, randomly according to its weight. This process generates a random tree of multi-graph star.

Now we assume that the lemma holds for all V_1' with $|V_1'| < k$. Let $V_1 = \{v_1, ..., v_k\}$. The key thing to note is that when V_1 is an independent set, we can write

$$\textsc{Sc}\left(G, V_2\right) = G\left[V_2\right] + K\left(v_1\right) + \ldots + K\left(v_k\right),$$

and

$$\textsc{Sc}\left(G, V_2 \cup v_k\right) = G\left[V_2 \cup v_k\right] + K\left(v_1\right) + \ldots + K\left(v_{k-1}\right).$$

Therefore, by the same reasoning as above, we can take a random tree T' of the multi-graph $G[V_2] + K(v_1) + ... + K(v_k)$ and map it to a tree on $G[V_2 \cup v_k] + K(v_1) + ... + K(v_{k-1}) = \textsc{Sc}\left(G, V_2 \cup v_k\right)$ by our procedure ProlongateTree. We then apply our inductive hypothesis on the set $V_1 \setminus \{v_k\}$ to map $\textsc{Sc}(G, V_2 \cup v_k)$ to a tree of G by ProlongateTree, which implies the lemma.

□

We also remark that the running time of ProlongateTree can be reduced to $O(m \log n)$ using dynamic trees, which can be abstracted as a data structure supporting operations on rooted forests [112, 113]. We omit the details here as this does not bottleneck the running time.

With this procedure fixed, we can now show the overall guarantees of the exact algorithm.

Proof. of Lemma 2.8.1 Correctness follows immediately from Lemmas 2.8.2 and 2.8.5. The running time of ProlongateTree is $O(n^2)$ and contracting or deleting all edges contained in $G[V_1]$ takes $O(m)$ time. Note that in this new contracted graph, the vertex set containing V_1 is an independent set. Furthermore, computing the Schur complement takes $O(n^\omega)$ time, giving the running time recurrence

$$T\left(n\right) = 2T\left(n/2\right) + O\left(n^\omega\right) = O\left(n^\omega\right).$$

□

2.8.2 Fast Random Spanning Tree Sampling using Determinant Sparsification of Schur complement

Next, we note that the most expensive operation from the exact sampling algorithm from Section 2.8.1 was the Schur complement procedure. Accordingly, we will substitute in our sparse Schur complement procedure to speed up the running time.

However, this will add some complication in applying Line 5 of EXACTTREE. To address this, we need the observation that the SCHURSPARSE procedure can be extended to distinguish edges from the original graph, and the Schur complement in the multi-graph that it produces.

Lemma 2.8.6. *The procedure* SCHURSPARSE(G, V_1, δ) *given in Algorithm 4 can be modified to record whether an edge in its output,* H^{V_1} *is a rescaled copy of an edge from the original induced subgraph on* V_1*,* $G[V_1]$*, or one of the new edges generated from the Schur complement,* $G_{SC}(V_1)$*.*

Proof. The edges for H^{V_1} are generated by the random walks via SAMPLEEDGESCHUR(G, V_1), whose pseudocode is given in Algorithm 3. Each of these produces a walk between two vertices in V_1, and such a walk belongs to $G[V_1]$ if it is length 1, and $G_{SC}(V_1)$ otherwise. □

We can now give our algorithm for generating random spanning trees and prove the guarantees that lead to the main result from Theorem 2.2.3.

Note that the splitting on Line 7 is mapping T_1 first back to a tree on a the sparsified multi-graph of SC (G, V_1): where the rescaled edges that originated from $G[V_1]$ are tracked separately from the edges that arise from new edges involving random walks that go through vertices in V_2.

The desired runtime will follow equivalently to the analysis of the determinant algorithm in Section 2.7 as we are decreasing δ proportionally to the number of vertices. It remains to

Algorithm 9: APPROXTREE$(G, \delta, \overline{n})$ Take a graph and output a tree randomly from a distribution δ-close to the w-uniform distribution

Input: Graph G, error parameter δ, and initial number of vertices $\overline{n}$

Output: A tree randomly generated from a distribution δ-close to the w-uniform distribution of G

1 $V_2 \leftarrow$ ALMOSTINDEPENDENT$(G, \frac{1}{10})$; {Via Lemma 2.6.2}

2 $H_1 \leftarrow$ SCHURSPARSE$(G, V_1, \delta \cdot |V_1|/\overline{n}))$, while tracking whether the edge is from $G[V_1]$ via the modifications from Lemma 2.8.6 ;

3 $T_1 =$ APPROXTREE$(H_1, \delta, \overline{n})$;

4 $G' \leftarrow G$;

5 **for** *each* $e \in T_1$ **do**

6 **if** RAND$[0, 1] \leq w_e^{ori}(G_1)/w_e(G_1)$ **then** {$w_e^{ori}(G_1)$ is calculated using the weights tracked from Line 2);

7 $G' \leftarrow G'/\{e\}$;

8 $T \leftarrow T \cup \{e\}$;

9 Delete all edges between (remaining) vertices in V_1 in G', $G' \leftarrow G' \setminus E(G'[V_1])$;

10 $H_2 \leftarrow$ SCHURSPARSE$(G', V_2, \delta \cdot |V_2|/\overline{n})$;

11 $T_2 =$ APPROXTREE(H_2, δ, n);

12 $T \leftarrow T \cup$ PROLONGATETREE$(G, V_1 \sqcup V_2, T_2)$;

13 Output T;

bound the distortion to the spanning tree distribution caused by the calls to SCHURSPARSE.

Bounds on this distortion will not follow equivalently to that of the determinant algorithm, which also substitutes SCHURSPARSE for exact Schur complements, due to the dependencies in our recursive structure. In particular, while the calls to SCHURSPARSE are independent, the graphs that they are called upon depend on the randomness in Line 6 and PROLONGATETREE, which more specifically, are simply the resulting edge contractions/deletions in previously visited vertex partitions within the recursion. Each subgraph SCHURSPARSE is called upon is additionally dependent on the vertex partitioning from ALMOSTINDEPENDENT.

The key idea to our proof will then be a layer-by-layer analysis of distortion incurred by SCHURSPARSE at each layer to the probability of sampling a *fixed* tree. By considering an alternate procedure where we consider exactly sampling a random spanning tree after some layer, along with the fact that our consideration is restricted to a *fixed* tree, this will allow us

to separate the randomness incurred by calls to SCHURSPARSE from the other sources of randomness mentioned above. Accordingly, we will provide the following definition.

Definition 2.8.7. For any $L \geq 0$, the **level-L truncated algorithm** is the algorithm given by modifying APPROXTREE$(G, \delta, \overline{n})$ so that all computations of sparsified Schur complements are replaced by exact calls to Schur complements (aka. SC (G, V_1) or SC (G', V_2))) after level l.

The tree distribution $\mathcal{T}^{(L)}$ is defined as the output of the level-L truncated algorithm.

Note that in particular, $\mathcal{T}^{(0)}$ is the tree distribution produced by EXACTTREE(G), or the w-uniform distribution; while $\mathcal{T}^{(O(\log n))}$ is the distribution outputted by APPROXTREE(G, δ).

The primary motivation of this definition is that we can separate the randomness between $\mathcal{T}^{(l)}$ and $\mathcal{T}^{(l+1)}$ by only the calls to SCHURSPARSE at level $l+1$, which will ultimately give the following lemma that we prove at the end of this section

Lemma 2.8.8. *For an invocation of* APPROXTREE *on a graph G with variance bound δ, for any layer $L > 0$, we have*

$$d_{TV}\left(\mathcal{T}^{(L-1)}, \mathcal{T}^{(L)}\right) \leq O(\sqrt{\delta}).$$

To begin, we consider the differences between $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$ and the probability of sampling a fixed tree $\widehat{T}$ on a recursive call on G. The most crucial observation is that the two recursive calls to APPROXTREE$(G_1, \delta, \overline{n})$ and APPROXTREE$(G_2, \delta, \overline{n})$ can be viewed as independent:

Claim 2.8.9. *For a call to* APPROXTREE$(G, \delta, \overline{n})$ *(Algorithm 9) to return $\widehat{T}$, there is only one possible choice of G' as generated via Lines 4 to 9.*

Proof. Note that the edges removed from Line 7 are precisely the edges in T with both endpoints contained in V_1, $E(T[V_1])$. For a fixed $\widehat{T}$, this set is unique, so G' is unique as well. □

This allows us to analyze a truncated algorithm by splitting the probabilities into those that occur at level l or above. Specifically, at the first level, this can be viewed as pairs of graphs $\textsc{Sc}(G, V_1)$ and $\textsc{Sc}(G, V_2)$ along with the 'intended' trees from them:

Definition 2.8.10. We define the level-one probabilities of returning a pair of trees T_1 and T_2 that belong a pair of graphs G_1, G_2,

$$p^{(\leq 1)}\left((G, G_1, G_2), (T_1, T_2), \widehat{T}\right).$$

as the product of:

1. The probability (from running ALMOSTINDEPENDENT) that G is partitioned into $V_1 \sqcup V_2$ so that $\textsc{Sc}(G, V_1) = G_1$ and $\textsc{Sc}(G', V_2) = G_2$, where G' is G with the edges $T \cap G[V_1]$ contracted and all other edges in $G[V_1]$ are deleted.

2. The probability that T_1 is mapped to $\widehat{T}[V_1]$ in Line 6.

3. The probability that T_2 is mapped to $\widehat{T}/\widehat{T}[V_1]$ by the call to PROLONGATETREE on Line 12.

This definition then allows us to formalize the splitting of probabilities above and below level 1. More importantly, we note that if we instead call SCHURSPARSE to generate G_1 and G_2, this will not affect the level-one probability because (1) both the calls to ALMOSTINDEPENDENT and PROLONGATETREE do not depend on G_1 and G_2, and (2) we can consider T_1 to be drawn from the multi-graph of G_1 where we track which edges are from the original graph and which were generated by the Schur complement.

Consequently, the only difference between the distributions $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$ will be the distortion of drawing T_1 and T_2 from G_1 and G_2 vs the sparsified version of G_1 and G_2. This handling of sparsifiers of the Schur complements is further simplified with by the following observation:

Claim 2.8.11. *The output of* SCHURSPARSE(G, V', δ) *is identical to the output of*

$$\text{IDEALSPARSIFY}\left(\text{SC}\left(G, V'\right), \widetilde{\boldsymbol{\tau}}, n^2\delta^{-1}\right),$$

for some set of 1.1*-approximate statistical leverage scores of* SC (G, V')*,* $\widetilde{\boldsymbol{\tau}}$*.*

This can be seen by revisiting the Schur complement sparsification and rejection sampling algorithms from Section 2.6 and 2.5.3 which show that this statement also extends to the approximate Schur complements produced on lines 2 and 10 of Algorithm 9.

This means we can let $\mathcal{H}_1$ and $\mathcal{H}_2$ denote the distribution produced by IDEALSPARSIFY on G_1 and G_2 respectively.

Lemma 2.8.12. *There exists a collection of graphs and tree pairs* $(\vec{\mathcal{G}}, \vec{\mathcal{T}})^{\leq 1}$ *such that for any tree* $\widehat{T}$*, with the probabilities given above in Definition 2.8.10 we have:*

$$\boldsymbol{Pr}^{\mathcal{T}^{(0)}}\left(\widehat{T}\right) = \sum_{((G,G_1,G_2),(T_1,T_2))\in(\mathcal{G},\mathcal{T})^{(\leq 1)}} p^{(\leq 1)}\left((G, G_1, G_2), (T_1, T_2), \widehat{T}\right) \cdot \boldsymbol{Pr}^{G_1}(T_1) \cdot \boldsymbol{Pr}^{G_2}(T_2).$$

and

$$\boldsymbol{Pr}^{\mathcal{T}^{(1)}}\left(\widehat{T}\right) = \sum_{((G,G_1,G_2),(T_1,T_2))\in(\mathcal{G},\mathcal{T})^{(\leq 1)}} p^{(\leq 1)}\left((G, G_1, G_2), (T_1, T_2), \widehat{T}\right) \cdot \mathbb{E}\left[H_1 \in \mathcal{H}_1\right] \boldsymbol{Pr}^{H_1}(T_1) \cdot \mathbb{E}\left[H_2 \in \mathcal{H}_2\right] \boldsymbol{Pr}^{G_2}(T_2).$$

We can then in turn extend this via induction to multiple levels. It is important to note that in comparing the distributions $\mathcal{T}^{(L-1)}$ and $\mathcal{T}^{(L)}$ for $L \geq 1$ both will make calls to IDEALSPARSIFY through level L. We will then need to additionally consider the possible graphs generated by sparsification through level L, then restrict to the corresponding exact graphs at level $L + 1$.

Definition 2.8.13. We will use $\vec{\mathcal{G}}^{(\leq L)}, \vec{\mathcal{T}}^{(L)}$ to denote a sequence of graphs on levels up to $L-1$, plus the peripheral exact Schur complements on level L, along with the spanning trees generated on these peripheral graphs.

As these graphs and trees can exist on different vertex sets, we will use $(\vec{\mathcal{G}}, \vec{\mathcal{T}})^{(\leq L)}$ to denote the set of graph/tree pairs that are on the same set of vertices. For a sequence of graphs $\vec{G}^{\leq L}$ and a sequence of trees on their peripherals, $\vec{T}^{L}$, we will use

$$p^{(\leq L)}\left(\vec{G}^{(\leq L)}, \vec{T}^{(L)}, \widehat{T}\right)$$

to denote the product of the probabilities of the level-by-level vertex split and resulting trees mapping back correctly as defined in Definition 2.8.10, times the probabilities that the subsequent graphs are generated as sparsifiers of the ones above

Furthermore, we will use $\vec{G}^{(L)}$ to denote just the peripheral graphs, and $\vec{\mathcal{H}}(\vec{G}^{(L)})$ to denote the product distribution over sparsifiers of these graphs, and $\vec{H}^{(L)}$ to denote one particular sequence of such sparsifiers on this level. We can also define the probabilities of trees being picked in a vector-wise sense:

$$\boldsymbol{Pr}^{\vec{G}^{(L)}}\left(\vec{T}^{(L)}\right) \stackrel{\text{def}}{=} \prod_j \boldsymbol{Pr}^{\vec{G}_j^{(L)}}\left(\vec{T}_j^{(L)}\right), \qquad \boldsymbol{Pr}^{\vec{H}^{(L)}}\left(\vec{T}^{(L)}\right) \stackrel{\text{def}}{=} \prod_j \boldsymbol{Pr}^{\vec{H}_j^{(L)}}\left(\vec{T}_j^{(L)}\right).$$

Applying Lemma 2.8.12 inductively then allows us to extend this to multiple levels.

Corollary 2.8.14. *There exists a collection of graphs and tree pairs $(\vec{\mathcal{G}}, \vec{\mathcal{T}})^{(\leq L)}$ such that for any tree $\widehat{T}$ we have:*

$$\boldsymbol{Pr}^{\mathcal{T}^{(L-1)}}\left(\widehat{T}\right) = \sum_{\left(\vec{G}^{(\leq L)}, \vec{T}^{(L)}\right) \in (\mathcal{G}, \mathcal{T})^{(\leq L)}} p^{(\leq L)}\left(\vec{G}^{(\leq L)}, \vec{T}^{(L)}, \widehat{T}\right) \cdot \boldsymbol{Pr}^{\vec{G}^{(L)}}\left(\vec{T}^{(L)}\right),$$

and

$$\boldsymbol{Pr}^{\mathcal{T}^{(L)}}\left(\widehat{T}\right) = \sum_{\left(\vec{G}^{(\leq L)},\vec{T}^{(L)}\right)\in(\mathcal{G},\mathcal{T})^{(\leq L)}} p^{(\leq L)}\left(\vec{G}^{(\leq L)},\vec{T}^{(L)},\widehat{T}\right) \cdot \mathbb{E}\left[\vec{H}^{(L)}\sim\vec{\mathcal{H}}\left(G^{(L)}\right)\right]\boldsymbol{Pr}^{\vec{H}^{(L)}}\left(\vec{T}^{(L)}\right).$$

This reduces our necessary proof of bounding the total variation distance between $\mathcal{T}^{(L-1)}$ and $\mathcal{T}^{(L)}$ to examining the difference between

$$\boldsymbol{Pr}^{\vec{G}^{(L)}}\left(\vec{T}^{(L)}\right) \qquad \text{and} \qquad \mathbb{E}\left[\vec{H}^{(L)}\sim\vec{\mathcal{H}}\left(G^{(L)}\right)\right]\boldsymbol{Pr}^{\vec{H}^{(L)}}\left(\vec{T}^{(L)}\right).$$

Recalling the definition of $\boldsymbol{Pr}^{\vec{H}^{(L)}}(\vec{T}^{(L)})$: we have that the inverse of each probability in the expectation is

$$\boldsymbol{Pr}^{\vec{H}_j^{(L)}}\left(\vec{T}_j^{(L)}\right)^{-1} = \frac{\mathcal{T}_{\vec{H}_j^{(L)}}}{w^{\vec{H}_j^{(L)}}\left(\vec{T}_j^{(L)}\right)},$$

and we have concentration bounds for the total trees in $\vec{H}_j^{(L)}$. However, it is critical to note that this probability is 0 (and cannot be inverted) when $\vec{T}_j^{(L)}$ is not contained in $\vec{H}_j^{(L)}$ for some j.

This necessitates extending our concentration bounds to random graphs where we condition upon a certain tree remaining in the graph. This will be done in the following Lemma, proven in Section 2.9, and we recall that we set s such that $\delta = O(\frac{n^2}{s})$ in SCHURSPARSE.

Lemma 2.8.15. *Let G be a graph on n vertices and m edges, $\widetilde{\boldsymbol{\tau}}$ be an 1.1-approximate estimates of leverage scores, s be a sample count such that $s \geq 4n^2$ and $m \geq \frac{s^2}{n}$. Let $\mathcal{H}$ denote the distribution over the outputs of* IDEALSPARSIFY$(G,\widetilde{\boldsymbol{\tau}},s)$*, and for a any fixed spanning $\widehat{T}$, let $\mathcal{H}|_T$ denote the distribution formed by conditioning on the graph containing $\widehat{T}$. Then we have:*

$$\mathbb{P}\left[H\sim\mathcal{H}\right]\widehat{T}\subseteq H^{-1}\cdot\mathbb{E}\left[H|_{\widehat{T}}\sim\mathcal{H}_{|_{\widehat{T}}}\right]\boldsymbol{Pr}^{H|_{\widehat{T}}}\left(\widehat{T}\right)^{-1} = \left(1\pm O\left(\frac{n^2}{s}\right)\right)\boldsymbol{Pr}^{G}\left(\widehat{T}\right)^{-1},$$

and

$$\mathbb{P}\left[H \sim \mathcal{H}\right] \widehat{T} \subseteq H^{-2} \cdot \mathbf{Var}\left[H|_{\widehat{T}} \sim \mathcal{H}_{|_{\widehat{T}}}\right] \boldsymbol{Pr}^{H|_{\widehat{T}}}\left(\widehat{T}\right)^{-1} \leq O\left(\frac{n^2}{s}\right) \boldsymbol{Pr}^{G}\left(\widehat{T}\right)^{-2}.$$

Due to the independence of each call to IDEALSPARSIFY, we can apply these concentration bounds across the product

$$\boldsymbol{Pr}^{\vec{H}^{(L)}}\left(\vec{T}^{(L)}\right) = \prod_j \boldsymbol{Pr}^{\vec{H}_j^{(L)}}\left(\vec{T}_j^{(L)}\right)$$

and use the fact that δ decreases proportionally to vertex size in our algorithm:

Corollary 2.8.16. *For any sequence of peripheral graphs $\vec{G}^{(l)}$, with associated sparsifier distribution $\mathcal{H}^S$, and any sequence of trees $\vec{T}^{(L)}$ as defined in Definition 2.8.13 such that $\boldsymbol{Pr}^{\vec{G}^{(L)}}(\vec{T}^{(L)}) > 0$, we have*

$$\begin{aligned}\mathbb{P}\left[\vec{H}^{(L)} \sim \vec{\mathcal{H}}\left(G^{(L)}\right)\right] \boldsymbol{Pr}^{\vec{H}^{(L)}}\left(\vec{T}^{(L)}\right)^{-1} \\ \cdot \mathbb{E}\left[\vec{H}^{(L)} \sim \vec{\mathcal{H}}\left(G^{(L)}\right) \middle| \boldsymbol{Pr}^{\vec{H}^{(L)}}\left(\vec{T}^{(L)}\right) > 0\right] \boldsymbol{Pr}^{\vec{H}^{(L)}}\left(\vec{T}^{(L)}\right)^{-1} \\ = (1 \pm \delta)\, \boldsymbol{Pr}^{\vec{G}^{(L)}}\left(\vec{T}^{(L)}\right)^{-1},\end{aligned}$$

and

$$\begin{aligned}\mathbb{P}\left[\vec{H}^{(L)} \sim \vec{\mathcal{H}}\left(G^{(L)}\right)\right] \boldsymbol{Pr}^{\vec{H}^{(L)}}\left(\vec{T}^{(L)}\right)^{-2} \\ \cdot \mathbb{E}\left[\vec{H}^{(L)} \sim \vec{\mathcal{H}}\left(G^{(L)}\right) \middle| \boldsymbol{Pr}^{\vec{H}^{(L)}}\left(\vec{T}^{(L)}\right) > 0\right] \boldsymbol{Pr}^{\vec{H}^{(L)}}\left(\vec{T}^{(L)}\right)^{-2} \\ \leq (1 + \delta)\, \boldsymbol{Pr}^{\vec{G}^{(L)}}\left(\vec{T}^{(L)}\right)^{-2}.\end{aligned}$$

Proof. The independence of the calls to IDEALSPARSIFY, and the definition of

$$\boldsymbol{Pr}^{\vec{G}^{(L)}}\left(\vec{T}^{(L)}\right) \stackrel{\text{def}}{=} \prod_j \boldsymbol{Pr}^{\vec{G}_j^{(L)}}\left(\vec{T}_j^{(L)}\right), \qquad \boldsymbol{Pr}^{\vec{H}^{(L)}}\left(\vec{T}^{(L)}\right) \stackrel{\text{def}}{=} \prod_j \boldsymbol{Pr}^{\vec{H}_j^{(L)}}\left(\vec{T}_j^{(L)}\right).$$

Applying Lemma 2.8.15 to each call of IDEALSPARSIFY, where s was set such that $\delta/\overline{n} = \frac{n^2}{s}$ gives gives that the total error bounded by

$$\exp\left(\sum_j \frac{\left|V\left(G^{(L)}\right)\right|}{\overline{n}}\right),$$

and the bound then follows form the total size of each level of the recursion being $O(\bar{n})$. $\square$

It then remains to use concentration bounds on the inverse of the desired probability to bound the total variation distance, which can be done by the following lemma which can be viewed as an extension of Lemma 2.4.7, and is also proven in Section 2.10.

Lemma 2.8.17. *Let $\mathcal{U}$ be a distribution over a universe of elements, u, each associated with random variable P_u such that*

$$\mathbb{E}\left[u \sim \mathcal{U}\right] \mathbb{E}\left[\right] P_u = 1,$$

and for each P_u we have

1. $P_u \geq 0$, *and*
2. $\mathbb{P}\left[\right] P_u > 0^{-1} \cdot \mathbb{E}\left[p \sim P_u \,|p > 0\right] p^{-1} = 1 \pm \delta$, *and*
3. $\mathbb{P}\left[\right] P_u > 0^{-2} \mathbb{E}\left[p \sim P_u \,|p > 0\right] p^{-2} \leq 1 + \delta$,

then

$$\mathbb{E}\left[u \sim \mathcal{U}\right] |1 - \mathbb{E}\left[\right] P_u| \leq O\left(\sqrt{\delta}\right).$$

To utilize this lemma, we observe that the values

$$p^{(\leq L)}\left(\vec{G}^{(\leq L)}, \vec{T}^{(L)}, \widehat{T}\right) \cdot \boldsymbol{Pr}^{\vec{G}^{(L)}}\left(\vec{T}^{(L)}\right)$$

forms a probability distribution over tuples $\vec{G}^{(\leq L)}, \vec{T}^{(L)}, \widehat{T}$, while the distribution $\mathcal{H}(\vec{G}^{(L)})$, once rescaled, can play the role of P_u. Decoupling the total variation distance per tree into

the corresponding terms on pairs of $\vec{G}^{(\leq L)}, \vec{T}^{(L)}$ then allows us to bound the overall total variation distance between $\mathcal{T}^{(L-1)}$ and $\mathcal{T}^{(L)}$.

Proof of Lemma 2.8.8. By the definition of total variation distance

$$d_{TV}\left(\mathcal{T}^{(L-1)}, \mathcal{T}^{(L)}\right) = \sum_{\widehat{T}} \left| \boldsymbol{Pr}^{\mathcal{T}^{(L-1)}}\left(\widehat{T}\right) - \boldsymbol{Pr}^{\mathcal{T}^{(L)}}\left(\widehat{T}\right)\right|.$$

By Corollary 2.8.14 and triangle inequality we can then upper bound this probability by

$$\begin{aligned} d_{TV}\left(\mathcal{T}^{(L-1)}, \mathcal{T}^{(L)}\right) \leq \sum_{\widehat{T}} \sum_{\left(\vec{G}^{(\leq L)}, \vec{T}^{(L)}\right) \in (\mathcal{G}, \mathcal{T})^{(\leq L)}} & p^{(\leq L)}\left(\vec{G}^{(\leq L)}, \vec{T}^{(L)}, \widehat{T}\right) \\ & \cdot \left| \boldsymbol{Pr}^{\vec{G}^{(L)}}\left(\vec{T}^{(L)}\right) - \mathbb{E}\left[\vec{H}^{(L)} \sim \vec{\mathcal{H}}\left(G^{(L)}\right)\right] \boldsymbol{Pr}^{\vec{H}^{(L)}}\left(\vec{T}^{(L)}\right)\right|. \end{aligned}$$

The scalar $p^{(\leq L)}(\vec{G}^{(\leq L)}, \vec{T}^{(L)}, \widehat{T})$ is crucially the same for each, and the inner term in the summation is equivalent to

$$\begin{aligned} |p^{(\leq L)}\left(\vec{G}^{(\leq L)}, \vec{T}^{(L)}, \widehat{T}\right) & \cdot \boldsymbol{Pr}^{\vec{G}^{(L)}}\left(\vec{T}^{(L)}\right) \\ & - p^{(\leq L)}\left(\vec{G}^{(\leq L)}, \vec{T}^{(L)}, \widehat{T}\right) \cdot \mathbb{E}\left[\vec{H}^{(L)} \sim \vec{\mathcal{H}}\left(G^{(L)}\right)\right] \boldsymbol{Pr}^{\vec{H}^{(L)}}\left(\vec{T}^{(L)}\right)| \end{aligned}$$

Our goal is to use Lemma 2.8.17 where $\mathcal{U}$ here is the distribution over tuples $(\vec{G}^{(L)}, \vec{T}^{(L)}, \widehat{T})$ with density equaling:

$$p^{(\leq L)}\left(\vec{G}^{(\leq L)}, \vec{T}^{(L)}, \widehat{T}\right) \cdot \boldsymbol{Pr}^{\vec{G}^{(L)}}\left(\vec{T}^{(L)}\right),$$

and P_u is the distribution over the corresponding value of $\mathcal{H}(\vec{G}^{(L)})$, with the same density, and values equaling to:

$$\boldsymbol{Pr}^{\vec{G}^{(L)}}\left(\vec{T}^{(L)}\right)^{-1} \boldsymbol{Pr}^{\vec{H}^{(L)}}\left(\vec{T}^{(L)}\right).$$

Note that the fact that each $\vec{T}^L$ maps back to some tree $\hat{T}$ imply that $\mathcal{U}$ is a distribution, as well as $\mathbb{E}\left[u \sim \mathcal{U}\right] \mathbb{E}\left[\right] P_u = 1$. A rescaled version of Corollary 2.8.16 then gives the required conditions for Lemma 2.8.17, which in turn gives the overall bound.

□

Proof of Theorem 2.2.3. The running time follows the same way as the analysis of the determinant estimation algorithm in the Proof of Theorem 2.2.2 at the end of Section 2.7.

For correctness, the total variation distance bound is implied by appropriately setting δ, and then invoking the per-layer bound from Lemma 2.8.8. Note that factors of $\log n$ are absorbed by the $\widetilde{O}$ notation.

Finally, note that for simplicity our analysis of total variation distance does not account for the failure probability of Lemma 2.3.4. To account for these, we can simply use the fact that only $O(n \log n)$ calls to SCHURSPARSE are made. Hence, the probability of any call failing is polynomially small, which can be absorbed into the total variation distance. □

2.9 Conditional Concentration Bounds

In this section, we extend our concentration bounds to conditioning on a certain tree being in the sampled graph, specifically with the goal of proving Lemma 2.8.15. By edge splitting arguments similar to those in Section 2.5.2, it suffices to analyze the case where all edges have about the same leverage score.

Lemma 2.9.1. *Let G be a graph on n vertices and m edges such that all edges have statistical leverage scores $\overline{\boldsymbol{\tau}}_e \leq \frac{2n}{m}$, and s be a sample count such that $s \geq 4n^2$ and $m \geq \frac{s^2}{n}$. Let H be a subgraph containing s edges picked at random without replacement, and let $\mathcal{H}$ denote this distribution over subgraphs on s edges. Furthermore for any fixed spanning tree, $\widehat{T}$, let $\mathcal{H}|_T$ denote the distribution induced by those in $\mathcal{H}$ that contain $\widehat{T}$, and use $H|_{\widehat{T}}$ to*

denote such a graph, then

$$\mathbb{P}\left[H \sim \mathcal{H}\right] \widehat{T} \subseteq H^{-1} \cdot \mathbb{E}\left[H|_{\widehat{T}} \sim \mathcal{H}_{|_{\widehat{T}}}\right] \boldsymbol{Pr}^{H|_{\widehat{T}}}\left(\widehat{T}\right)^{-1} = \left(1 \pm O\left(\frac{n^2}{s}\right)\right) \boldsymbol{Pr}^{G}\left(\widehat{T}\right)^{-1},$$

and

$$\mathbb{P}\left[H \sim \mathcal{H}\right] \widehat{T} \subseteq H^{-2} \cdot \mathbf{Var}\left[H|_{\widehat{T}} \sim \mathcal{H}_{|_{\widehat{T}}}\right] \boldsymbol{Pr}^{H|_{\widehat{T}}}\left(\widehat{T}\right)^{-1} \leq O\left(\frac{n^2}{s}\right) \boldsymbol{Pr}^{G}\left(\widehat{T}\right)^{-2}.$$

Note that the 'uniform leverage score' requirement here is not as strict as the analysis from Lemma 2.5.2. This is because we're eventually aiming for a bound of $s \approx n^2$ samples. This also means that constant factor leverage score approximations suffices for this routine.

The starting point of this proof is the observation that because we're doing uniform sampling, the only term in

$$\boldsymbol{Pr}^{H|_{\widehat{T}}}\left(\widehat{T}\right) = \frac{w^{H|_{\widehat{T}}}\left(\widehat{T}\right)}{\mathcal{T}_{H|_{\widehat{T}}}} = \frac{w^{G}\left(\widehat{T}\right)}{\mathcal{T}_{H|_{\widehat{T}}}}$$

that is dependent on $H|_{\widehat{T}}$ is $\mathcal{T}_{H|_{\widehat{T}}}$. The proof will then follow by showing concentration of this variable which will be done similarly to the concentration of $\mathcal{T}_H$ that was done in Section 2.4 and 2.5.

The primary difficulty of extending the proof will come from the fact that trees will have different probabilities of being in the sampled graph depending on how many edges they share with $\widehat{T}$. Much of this will be dealt with by the assumption that $s \geq 4n^2$, which makes the exponential terms in the probabilities associated with a tree being in a sampled graph negligible. Additionally, this assumption implies that for any fixed tree $\widehat{T}$ the expected number of edges it shares with a random tree is close to 0. As a result, trees that intersect with $\widehat{T}$ will have negligible contributions, and our analysis can follow similarly to that in Section 2.4 and 2.5.

We further note that due to the larger sample count of $s \geq 4n^2$, the concentration bounds

in this section will also hold, and would in fact be slightly simpler to prove, if the edges were sampled independently with probability s/m. We keep our assumption of sampling s edges globally without replacement though in order to avoid changing our algorithm, and the analysis will not require much additional work.

The section will be organized as follows: In Section 2.9.1 we give upper and lower bounds on the expectation of $\mathcal{T}_{H|_{\widehat{T}}}$. In Section 2.9.2 we give an upper bound on the variance of $\mathcal{T}_{H|_{\widehat{T}}}$. In Section 2.9.3 we combine the bounds from the previous two sections to prove Lemma 2.9.1.

2.9.1 Upper and Lower Bounds on Conditional Expectation

In order to prove upper and lower bounds on $\mathbb{E}\left[H|_{\widehat{T}}\right]\mathcal{T}_{H|_{\widehat{T}}}$, we will first give several helpful definitions, corollaries, and lemmas to assist in the proof. Our examination of $\mathbb{E}\left[H|_{\widehat{T}}\right]\mathcal{T}_{H|_{\widehat{T}}}$ will require approximations of $\mathbb{P}\left[H|_{\widehat{T}}\right]T \subseteq H|_{\widehat{T}}$, and, as we are now fixing $n-1$ edges and drawing $s-n+1$ edges from the remaining $m-n+1$ edges, each edge will now have probability $\frac{s-n+1}{m-n+1}$ of being in the sampled graph. We will denote this probability with

$$\widehat{p} \stackrel{\text{def}}{=} \frac{s-n+1}{m-n+1}.$$

It will often be easier to exchange $\widehat{p}$ for

$$p \stackrel{\text{def}}{=} \frac{s}{m},$$

the probability of a single edge being picked without the conditioning on $\widehat{T}$. The errors of doing so is governed by:

$$\left(1-\frac{n}{s}\right)p = \frac{s-n}{m} \leq \frac{s-n+1}{m-n+1} = \hat{p} \leq \frac{s}{m} = p. \tag{2.4}$$

We remark that these errors turn out to be acceptable even when $\widehat{p}$ is raised to the $O(n)$ power.

Furthermore, our assumption of $s \geq 4n^2$ implies that we expect a randomly chosen tree not to intersect with $\widehat{T}$. This will often implicitly show up in the form of the geometric series below, for which a bound is immediately implied by our assumption.

Lemma 2.9.2. *If $s \geq 4n^2$, then*

$$\sum_{k=1}^{\infty} \left(\frac{2n^2}{s}\right)^k = O\left(\frac{n^2}{s}\right).$$

The change in our sampling procedure will alter the formulation of $\mathbb{P}\left[H|_{\widehat{T}}\right] T \subseteq H|_{\widehat{T}}$, so we first want to write $\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}_{H|_{\widehat{T}}}$ in terms of values that we are familiar with while only losing small errors. Additionally, many of the exponential terms in the previous analysis will immediately be absorbed into approximation error by our assumption that $s \geq 4n^2$.

Lemma 2.9.3. *Let G be a graph on n vertices and m edges and s a value such that $m \geq \frac{s^2}{n}$, Fix some tree $\widehat{T} \in G$. For a random subset of $s \geq 4n^2$ edges containing $\widehat{T}$, $H|_{\widehat{T}} \supseteq \widehat{T}$, we have*

$$\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}_{H|_{\widehat{T}}} = \left(1 - O\left(\frac{n^2}{s}\right)\right) \sum_{k=0}^{n-1} p^{n-1-k} \sum_{T:\left|T \cap \widehat{T}\right| = k} w(T),$$

where $p = s/m$ is the probability of each edge being picked in the sample.

Proof. Given that all edges of $\widehat{T}$ are in $H|_{\widehat{T}}$, the remaining $s - n + 1$ edges are chosen uniformly from all $m - n + 1$ edges not in $\widehat{T}$. Accordingly, for any tree $T \in G$, the probability $\mathbb{P}\left[H|_{\widehat{T}}\right] T \subseteq H|_{\widehat{T}}$ is obtained by dividing the number of subsets of $s - n + 1$ edges that contain all edges in $T \setminus \widehat{T}$, against the number of subsets of $s - n + 1$ edges from $m - n + 1$:

$$\mathbb{P}\left[H|_{\widehat{T}}\right] T \subseteq H|_{\widehat{T}} = \binom{m - n + 1 - \left|T \setminus \widehat{T}\right|}{s - n + 1 - \left|T \setminus \widehat{T}\right|} / \binom{m - n + 1}{s - n + 1} = \frac{(s - n + 1)_{|T \setminus \widehat{T}|}}{(m - n + 1)_{|T \setminus \widehat{T}|}}.$$

Following the proof Lemma 2.4.1, this reduces to

$$\mathbb{P}\left[H|_{\widehat{T}}\right] T \subseteq H|_{\widehat{T}} = \widehat{p}^{|T\setminus\widehat{T}|} \exp\left(-\frac{\left|T \setminus \widehat{T}\right|^2}{2s} - O\left(\frac{n^3}{s^2}\right)\right),$$

which we can further reduce using the assumption of $s \geq 4n^2$ to:

$$\mathbb{P}\left[H|_{\widehat{T}}\right] T \subseteq H|_{\widehat{T}} = \left(1 - O\left(\frac{n^2}{s}\right)\right)\widehat{p}^{|T\setminus\widehat{T}|},$$

and in turn obtain via linearity of expectation:

$$\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}_{H|_{\widehat{T}}} = \left(1 - O\left(\frac{n^2}{s}\right)\right)\sum_{T} w(T)\widehat{p}^{|T\setminus\widehat{T}|}.$$

We then subdivide the summation based on the amount of edges in the intersection of T and $\widehat{T}$ and move our $\widehat{p}$ term inside the summation

$$\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}_{H|_{\widehat{T}}} = \left(1 - O\left(\frac{n^2}{s}\right)\right)\sum_{k=0}^{n-1}\widehat{p}^{n-1-k}\sum_{T:\, T\cap\widehat{T}=k} w(T).$$

Finally, we can use Equation 2.4 to replace $\widehat{p}$ by p because

$$1 \geq \left(1 - \frac{n}{s}\right)^n \geq \left(1 - \frac{2n^2}{s}\right)$$

where $n^2 s < 0.1$.

□

We will also require a strong lower bound of the expectation. The following lemma shows that most of the trees do not intersect with $\widehat{T}$. Restricting our consideration to such trees will be much easier to work in obtaining the lower bound on $\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}_{H|_{\widehat{T}}}$.

Lemma 2.9.4. *Let G be a graph on n vertices and m edges such that $m \geq 4n^2$ and all*

edges have statistical leverage scores $\leq \frac{2n}{m}$*. For any tree* $\widehat{T} \in G$*.*

$$\sum_{T:\left|T\cap\widehat{T}\right|=0} w(T) \geq \left(1 - O\left(\frac{n^2}{s}\right)\right)\mathcal{T}_G.$$

Proof. By definition, we can classify the trees by their intersection with $\widehat{T}$:

$$\mathcal{T}_G = \sum_{k=0}^{n-1} \sum_{T:\left|T\cap\widehat{T}\right|=k} w(T).$$

Consider each inner summation and further separating into each possible forest of $\widehat{T}$ with k edges gives:

$$\sum_{T:\left|T\cap\widehat{T}\right|=k} w(T) = \sum_{\substack{F\subseteq\widehat{T}\\|F|=k}} \sum_{\substack{T\\F=T\cap\widehat{T}}} w(T) \leq \sum_{\substack{F\subseteq\widehat{T}\\|F|=k}} \sum_{T:\,F\subseteq T} w(T).$$

Invoking Lemma 2.4.4 on the inner summation and the fact that there are $\binom{n-1}{k}$ forests of $\widehat{T}$ with k edges, gives an upper bound of

$$\sum_{T:\left|T\cap\widehat{T}\right|=k} w(T) \leq \binom{n-1}{k}\mathcal{T}_G\left(\frac{2n}{m}\right)^k \leq \mathcal{T}_G\left(\frac{2n^2}{m}\right)^k.$$

We will utilize this upper bound for all $k > 0$ and achieve a lower bound from rearranging our initial summation

$$\sum_{T:\left|T\cap\widehat{T}\right|=0} w(T) = \mathcal{T}_G - \sum_{k=1}^{n-1} \sum_{T:\left|T\cap\widehat{T}\right|=k} w(T) \geq \mathcal{T}_G\left(1 - \sum_{k=1}^{n-1}\left(\frac{2n^2}{m}\right)^k\right).$$

Applying the assumption of $m \geq 4n^2$ and Lemma 2.9.2 gives our desired result. □

With the necessary tools in place, we will now give upper and lower bounds on the expectation in terms of $\mathcal{T}_G p^{n-1}$, which we note is also a close approximation of $\mathbb{E}\left[H\right]\mathcal{T}_H$

by our assumption that $s \geq 4n^2$.

Lemma 2.9.5. *Let G be a graph on n vertices and m edges such that all edges have statistical leverage scores $\leq \frac{2n}{m}$, and let s be such that $m \geq \frac{s^2}{n}$. Fix some tree $\widehat{T} \in G$. For a random subset of $s \geq 4n^2$ edges that contain $\widehat{T}$, $H|_{\widehat{T}} \subseteq \widehat{T}$ we have:*

$$\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}_{H|_{\widehat{T}}} = \left(1 \pm O\left(\frac{n^2}{s}\right)\right) \mathcal{T}_G p^{n-1}.$$

Proof. We will first prove the upper bound. From Lemma 2.9.3 we have

$$\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}_{H|_{\widehat{T}}} \leq \sum_{k=0}^{n-1} p^{n-1-k} \sum_{T:\left|T \cap \widehat{T}\right|=k} w\left(T\right),$$

while a proof similar to Lemma 2.9.4 gives

$$\sum_{T:\left|T \cap \widehat{T}\right|=k} w(T) \leq \mathcal{T}_G \left(\frac{2n^2}{m}\right)^k.$$

Moving p^{n-1} outside the summation and substituting $\frac{s}{m}$ for p gives

$$\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}_{H|_{\widehat{T}}} \leq \mathcal{T}_G p^{n-1} \sum_{k=0}^{n-1} \left(\frac{2n^2}{s}\right)^k,$$

and applying Corollary 2.9.2 to upper bound the summation gives

$$\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}_{H|_{\widehat{T}}} \leq \left(1 + O\left(\frac{n^2}{s}\right)\right) \mathcal{T}_G p^{n-1}.$$

For the lower bound, we again first using Lemma 2.9.3 and then restrict to trees that do

not intersect $\widehat{T}$ using Lemma 2.9.4. Formally we have:

$$\mathbb{E}\left[H|_{\widehat{T}}\right]\mathcal{T}_{H|_{\widehat{T}}} = \left(1 - O\left(\frac{n^2}{s}\right)\right)\sum_{k=0}^{n-1} p^{n-1-k} \sum_{T:\left|T\cap\widehat{T}\right|=k} w\left(T\right)$$
$$\geq \left(1 - O\left(\frac{n^2}{s}\right)\right)p^{n-1} \sum_{T:\left|T\cap\widehat{T}\right|=0} w\left(T\right) \geq \left(1 - O\left(\frac{n^2}{s}\right)\right)p^{n-1}\mathcal{T}_G.$$

□

2.9.2 Upper Bound on Conditional Variance

The bound on variance is by upper bounding $\mathbb{E}\left[H|_{\widehat{T}}\right]\mathcal{T}^2_{H|_{\widehat{T}}}$ in a way similar to Lemma 2.4.6. Once again, the assumption of $s > 4n^2$ means the situation is simpler because the exponential term is negligible.

As with the proof of Lemma 2.4.6, we will often separate summations of pairs of trees based upon the number of edges in their intersection, then frequently invoke Lemma 2.4.4. However there will be more moving pieces in each summation due to intersections with $\widehat{T}$, so Lemma 2.9.7 proven later in this section, which is analogous to Lemma 2.4.5, will be much more involved.

Lemma 2.9.6. *Let G be a graph on n vertices and m edges such that all edges have statistical leverage scores $\leq \frac{2n}{m}$, and s a sample count such that $m \geq \frac{s^2}{n}$. For some tree $\widehat{T} \in G$, let $H|_{\widehat{T}}$ denote a random subset of s edges such that $\widehat{T} \subseteq H|_{\widehat{T}}$, then:*

$$\frac{\mathbb{E}\left[H|_{\widehat{T}}\right]\mathcal{T}^2_{H|_{\widehat{T}}}}{\mathbb{E}\left[H|_{\widehat{T}}\right]{\mathcal{T}_{H|_{\widehat{T}}}}^2} \leq \left(1 + O\left(\frac{n^2}{s}\right)\right).$$

Proof. By analogous reasoning to the proof in Lemma 2.9.3, for any pair of trees $T_1, T_2 \in G$

we have

$$\mathbb{P}\left[H|_{\widehat{T}}\right] T_1, T_2 \subseteq H|_{\widehat{T}} = \binom{m-n+1-\left|(T_1 \cup T_2) \setminus \widehat{T}\right|}{s-n+1-\left|(T_1 \cup T_2) \setminus \widehat{T}\right|} / \binom{m-n+1}{s-n+1}$$
$$= \frac{(s-n+1)_{\left|(T_1 \cup T_2) \setminus \widehat{T}\right|}}{(m-n+1)_{\left|(T_1 \cup T_2) \setminus \widehat{T}\right|}}.$$

As a consequence of Equation 2.4, specifically the bound $\frac{s-k}{m-k} \leq \frac{s}{m}$ when $k \geq 0$, we can obtain the upper bound

$$\mathbb{P}\left[H|_{\widehat{T}}\right] T_1, T_2 \subseteq H|_{\widehat{T}} \leq p^{\left|(T_1 \cup T_2) \setminus \widehat{T}\right|},$$

and in turn summing over all pairs of trees:

$$\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}^2_{H|_{\widehat{T}}} \leq \sum_{T_1, T_2} w\left(T_1\right) w\left(T_2\right) p^{\left|(T_1 \cup T_2) \setminus \widehat{T}\right|}.$$

We note that $|(T_1 \cup T_2) \setminus \widehat{T}| = |T_1 \setminus \widehat{T}| + |T_2 \setminus \widehat{T}| - |(T_1 \cap T_2) \setminus \widehat{T}|$. Furthermore, $|T_1 \setminus \widehat{T}| = n - 1 - |T_1 \cap \widehat{T}$, so we separate the summation as per usual by each possible size of $|T_1 \cap \widehat{T}|$ and $|T_2 \cap \widehat{T}|$, and bring the terms outside of the summation that only depend on these values.

$$\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}^2_{H|_{\widehat{T}}} \leq p^{2n-2} \sum_{k_1, k_2} p^{-k_1-k_2} \sum_{\substack{T_1, T_2 \\ \left|T_1 \cap \widehat{T}\right| = k_1 \\ \left|T_2 \cap \widehat{T}\right| = k_2}} w\left(T_1\right) w\left(T_2\right) p^{-(T_1 \cap T_2) \setminus \widehat{T}}.$$

In order to deal with the inner most summation we will need to again separate based on the

size of $|(T_1 \cup T_2) \setminus \widehat{T}|$, and we further note that $|(T_1 \cap T_2) \setminus \widehat{T}| = |(T_1 \setminus \widehat{T}) \cap (T_2 \setminus \widehat{T})|$:

$$\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}^2_{H|_{\widehat{T}}} \leq p^{2n-2} \sum_{k_1,k_2} p^{-k_1-k_2} \sum_{k=0}^{n-1} p^{-k} \sum_{\substack{T_1,T_2 \\ \left|T_1 \cap \widehat{T}\right|=k_1 \\ \left|T_2 \cap \widehat{T}\right|=k_2 \\ \left|(T_1 \setminus \widehat{T}) \cap (T_2 \setminus \widehat{T})\right|=k}} w\left(T_1\right) w\left(T_2\right).$$

The last term is bounded in Lemma 2.9.7, which is stated and proven immediately after this. Incorporating the resulting bound, and grouping the terms by the summations over k_1, k_2, and k respectively gives:

$$\begin{aligned}
\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}^2_{H|_{\widehat{T}}} &\leq p^{2n-2} \sum_{k_1,k_2} p^{-k_1-k_2} \sum_{k=0}^{n-1} p^{-k} \binom{m}{k} \binom{n}{k_1} \binom{n}{k_2} \left(\frac{2n}{m}\right)^{2k+k_1+k_2} \mathcal{T}^2_G \\
&= \mathcal{T}^2_G p^{2n-2} \left(\sum_{k_1=0}^{n-1} p^{-k_1} \binom{n}{k_1} \left(\frac{2n}{m}\right)^{k_1}\right) \\
&\quad \cdot \left(\sum_{k_2=0}^{n-1} p^{-k_2} \binom{n}{k_2} \left(\frac{2n}{m}\right)^{k_2}\right) \left(\sum_{k=0}^{n-1} p^{-k} \binom{m}{k} \left(\frac{2n}{m}\right)^{2k}\right).
\end{aligned}$$

We then plug in $\frac{s}{m}$ for p in each summation and use the very crude upper bound $\binom{a}{b} \leq a^b$:

$$\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}^2_{H|_{\widehat{T}}} \leq \mathcal{T}^2_G p^{2n-2} \left(\sum_{k_1=0}^{n-1} \left(\frac{2n^2}{s}\right)^{k_1}\right) \left(\sum_{k_2=0}^{n-1} \left(\frac{2n^2}{s}\right)^{k_2}\right) \left(\sum_{k=0}^{n-1} \left(\frac{2n^2}{s}\right)^{k}\right).$$

Lemma 2.9.2 then upper bounds each summation by $1 + O(n^2/s)$, giving

$$\mathbb{E}\left[H|_{\widehat{T}}\right] \mathcal{T}^2_{H|_{\widehat{T}}} \leq \left(1 + O\left(\frac{n^2}{s}\right)\right) \mathcal{T}^2_G p^{2n-2}.$$

□

It remains to prove the following bound on the number of of pairs of trees with a certain intersection size with $\widehat{T}$, and each other. The following Lemma is a generalization to Lemma 2.4.5, and is proven analogously using the negative correlation of edges in spanning

trees from Fact 2.4.3 and Lemma 2.4.4.

Lemma 2.9.7. *Let G be graph with m edges and n vertices such that every edges has leverage score $\leq \frac{2n}{m}$. For any tree $\widehat{T} \in G$ and any integers $k, k_1, k_2 \in [0, n-1]$,*

$$\sum_{\substack{T_1,T_2 \\ \left|T_1 \cap \widehat{T}\right|=k_1 \\ \left|T_2 \cap \widehat{T}\right|=k_2 \\ \left|(T_1 \setminus \widehat{T}) \cap (T_2 \setminus \widehat{T})\right|=k}} w\left(T_1\right) w\left(T_2\right) \leq \binom{m}{k}\binom{n}{k_1}\binom{n}{k_2}\left(\frac{2n}{m}\right)^{2k+k_1+k_2} \mathcal{T}_G^2.$$

Proof. We will first separate the summation over all possible forests F of size k that could be the intersection of $T_1 \setminus \widehat{T}$ and $T_2 \setminus \widehat{T}$:

$$\sum_{\substack{T_1,T_2 \\ \left|T_1 \cap \widehat{T}\right|=k_1 \\ \left|T_2 \cap \widehat{T}\right|=k_2 \\ \left|(T_1 \setminus \widehat{T}) \cap (T_2 \setminus \widehat{T})\right|=k}} w\left(T_1\right) w\left(T_2\right) = \sum_{\substack{F \subseteq E \\ |F|=k}} \sum_{\substack{T_1,T_2 \\ \left|T_1 \cap \widehat{T}\right|=k_1 \\ \left|T_2 \cap \widehat{T}\right|=k_2 \\ F=(T_1 \setminus \widehat{T}) \cap (T_2 \setminus \widehat{T})}} w\left(T_1\right) w\left(T_2\right).$$

We first consider the inner summation, and will relax the requirement to only needing

$$F \subseteq (T_1 \setminus \widehat{T}) \cap (T_2 \setminus \widehat{T}),$$

which we note is equivalent to $F \subseteq (T_1 \setminus \widehat{T})$ and $F \subseteq (T_2 \setminus \widehat{T})$. This then allows us to separate the summation again for a particular F into terms involving just T_1 and T_2:

$$\sum_{\substack{T_1,T_2 \\ \left|T_1 \cap \widehat{T}\right|=k_1 \\ \left|T_2 \cap \widehat{T}\right|=k_2 \\ F=\left(T_1 \setminus \widehat{T}\right) \cap \left(T_2 \setminus \widehat{T}\right)}} w\left(T_1\right) w\left(T_2\right) \leq \left(\sum_{\substack{T_1 : \left|T_1 \cap \widehat{T}\right|=k_1 \\ F \subseteq \left(T_1 \setminus \widehat{T}\right)}} w\left(T_1\right) \right) \left(\sum_{\substack{T_2 : \left|T_2 \cap \widehat{T}\right|=k_2 \\ F \subseteq \left(T_2 \setminus \widehat{T}\right)}} w\left(T_2\right) \right).$$

We further examine the first term in the product, and the second will follow equivalently. Once again, we will split the summation by all possible forests $\widehat{F}$ of $\widehat{T}$ with size k_1 that

$T_1 \setminus \widehat{T}$ could intersect in, and further relax to them only having to contain $\widehat{F}$.

$$\sum_{\substack{T_1 : \left|T_1 \cap \widehat{T}\right| = k_1 \\ F \subseteq \left(T_1 \setminus \widehat{T}\right)}} w\left(T_1\right) \leq \sum_{\substack{\widehat{F} \subseteq \widehat{T} \\ \left|\widehat{F}\right| = k_1}} \sum_{\substack{T_1 \\ \widehat{F} \subseteq \left(T_1 \cap \widehat{T}\right) \\ F \subseteq \left(T_1 \setminus \widehat{T}\right)}} w\left(T_1\right).$$

Since $T_1 \cap \widehat{T}$ and $T_1 \setminus \widehat{T}$ are disjoint, we can restrict to $\widehat{F}$ that are disjoint from F, as well as relaxing to requiring $\left(\widehat{F} \cup F\right) \subseteq T_1$ (instead of $\widehat{F} \subseteq \left(T_1 \cap \widehat{T}\right)$ and $F \subseteq \left(T_1 \setminus \widehat{T}\right)$):

$$\sum_{\substack{T_1 : \left|T_1 \cap \widehat{T}\right| = k_1 \\ F \subseteq \left(T_1 \setminus \widehat{T}\right)}} w\left(T_1\right) \leq \sum_{\substack{\widehat{F} \subseteq \widehat{T} \\ \left|\widehat{F}\right| = k_1 \\ \left(\widehat{F} \cap F\right) = \emptyset}} \sum_{\left(\widehat{F} \cup F\right) \subseteq T} w\left(T\right).$$

The assumption of $\widehat{F}$ and F being disjoint means their union must have exactly $k + k_1$ edges. We can then apply Lemma 2.4.4 to the inner summation and use the fact that there are at most $\binom{n-1}{k_1}$ sets $\widehat{F}$ to achieve the upper bound

$$\sum_{\substack{T_1 : \left|T_1 \cap \widehat{T}\right| = k_1 \\ F \subseteq \left(T_1 \setminus \widehat{T}\right)}} w\left(T_1\right) \leq \binom{n}{k_1} \left(\frac{2n}{m}\right)^{k+k_1} \mathcal{T}_G.$$

Similarly, we can also obtain

$$\sum_{\substack{T_2 : \left|T_2 \cap \widehat{T}\right| = k_2 \\ F \subseteq \left(T_2 \setminus \widehat{T}\right)}} w\left(T_2\right) \leq \binom{n}{k_2} \left(\frac{2n}{m}\right)^{k+k_2} \mathcal{T}_G,$$

which, along with the fact that there are $\binom{m}{k}$ edge sets F of size k, gives our desired bound. $\square$

2.9.3 Concentration of Inverse Probabilities

We now complete a proof of Lemma 2.9.1 using the concentration results on the number of trees in a sampled graph, conditioned upon a certain tree being contained in the graph.

Proof of Lemma 2.9.1. The definition of

$$\boldsymbol{Pr}^{H|_{\widehat{T}}}\left(\widehat{T}\right)^{-1} = \frac{\mathcal{T}_{H|_{\widehat{T}}}}{w(\widehat{T})}$$

and Lemma 2.4.1 give

$$\mathbb{P}\left[H \sim \mathcal{H}\right]\widehat{T} \subseteq H^{-1} \cdot \mathbb{E}\left[H|_{\widehat{T}} \sim \mathcal{H}_{|_{\widehat{T}}}\right]\boldsymbol{Pr}^{H|_{\widehat{T}}}\left(\widehat{T}\right)^{-1}$$
$$= \left(\frac{1}{p}\right)^{n-1}\exp\left(\frac{n^2}{2s} + O\left(\frac{n^3}{s^2}\right)\right)\frac{\mathbb{E}\left[H|_{\widehat{T}} \sim \mathcal{H}_{|_{\widehat{T}}}\right]\mathcal{T}_{H|_{\widehat{T}}}}{w\left(\widehat{T}\right)}.$$

Our condition of $s \geq 4n^2$ allows us to bound the term $\exp(n^2/(2s) + O(n^3/s^2))$ by $(1 + O(n^2/s))$, and incorporating our approximation of $\mathbb{E}\left[H|_{\widehat{T}} \sim \mathcal{H}_{|_{\widehat{T}}}\right]\mathcal{T}_{H|_{\widehat{T}}}$ from Lemma 2.9.5 gives

$$\mathbb{P}\left[H \sim \mathcal{H}\right]\widehat{T} \subseteq H^{-1} \cdot \mathbb{E}\left[H|_{\widehat{T}} \sim \mathcal{H}_{|_{\widehat{T}}}\right]\boldsymbol{Pr}^{H|_{\widehat{T}}}\left(\widehat{T}\right)^{-1} = \left(1 \pm O\left(\frac{n^2}{s}\right)\right) \cdot \frac{\mathcal{T}_G}{w\left(\widehat{T}\right)},$$

and the definition of $\boldsymbol{Pr}^{G}\left(\widehat{T}\right)^{-1}$ implies the bounds on expectation.

For the variance bound, we use the identity

$$\mathbf{Var}\left[H|_{\widehat{T}} \sim \mathcal{H}_{|_{\widehat{T}}}\right]\boldsymbol{Pr}^{H|_{\widehat{T}}}\left(\widehat{T}\right)^{-1} =$$
$$\mathbb{E}\left[H|_{\widehat{T}} \sim \mathcal{H}_{|_{\widehat{T}}}\right]\boldsymbol{Pr}^{H|_{\widehat{T}}}\left(\widehat{T}\right)^{-2} - \mathbb{E}\left[H|_{\widehat{T}} \sim \mathcal{H}_{|_{\widehat{T}}}\right]\boldsymbol{Pr}^{H|_{\widehat{T}}}\left(\widehat{T}\right)^{-1^2},$$

which by the definition

$$\boldsymbol{Pr}^{H|_{\widehat{T}}}\left(\widehat{T}\right)^{-1} = \frac{\mathcal{T}_{H|_{\widehat{T}}}}{w(\widehat{T})}$$

reduces to

$$\begin{aligned}&\mathbf{Var}\left[H|_{\widehat{T}} \sim \mathcal{H}_{|_{\widehat{T}}}\right] \boldsymbol{Pr}^{H|_{\widehat{T}}}\left(\widehat{T}\right)^{-1} \\ &\qquad = \frac{\mathbb{E}\left[H|_{\widehat{T}} \sim \mathcal{H}_{|_{\widehat{T}}}\right] \mathcal{T}^2_{H|_{\widehat{T}}} - \mathbb{E}\left[H|_{\widehat{T}} \sim \mathcal{H}_{|_{\widehat{T}}}\right] \mathcal{T}_{H|_{\widehat{T}}}{}^2}{w\left(\widehat{T}\right)^2} \leq O\left(\frac{n^2}{s}\right) \cdot \frac{\mathcal{T}_G^2 p^{2n-2}}{w(\widehat{T})^2},\end{aligned}$$

where the last inequality is from incorporating Lemmas 2.9.5 and 2.9.6. Applying Lemma 2.4.1, and once again using the condition of $s \geq 4n^2$ to bound

$$\exp\left(\frac{n^2}{2s} + O(\frac{n^3}{s^2}\right) \leq \left(1 + O\left(\frac{n^2}{s}\right)\right) \leq O\left(1\right)$$

gives:

$$\mathbb{P}\left[H \sim \mathcal{H}\right] \widehat{T} \subseteq H^{-2} \cdot \mathbf{Var}\left[H|_{\widehat{T}} \sim \mathcal{H}_{|_{\widehat{T}}}\right] \boldsymbol{Pr}^{H|_{\widehat{T}}}\left(\widehat{T}\right)^{-1} \leq O\left(\frac{n^2}{s}\right) \cdot \frac{\mathcal{T}_G^2}{w(\widehat{T})^2},$$

and the variance bound follows from the definition of $\boldsymbol{Pr}^G\left(\widehat{T}\right)^{-1}$. □

2.10 Bounding Total Variation Distance

In this section we will first bound the total variation distance between drawing a tree from the w-uniform distribution of G, and uniformly sampling s edges, H, from G, then drawing a tree from the w-uniform distribution of H. The first bound will only be based on a concentration for the number of trees in H, and will give the $\tilde{O}(n^{13/6})$ time algorithm for sampling spanning trees from Corollary 2.4.8.

Next we will give a more general bound on the total variation distance between two distributions based on concentration of inverse probabilities. The resulting Lemma 2.8.17 is used for proving the bound on total variation distance in the recursive algorithm given in Section 2.8. However, as this bound requires a higher sample count of about n^2, the direct derivation of TV distances from concentration bounds is still necessary for uses of the

$\tilde{O}(n^{1.5})$ edge sparsifier in Corollary 2.4.8.

2.10.1 Simple Total Variation Distance Bound from Concentration Bounds

We give here a proof of total variation distance being bounded based on the concentration of spanning trees in the sampled graph.

Proof. (of Lemma 2.4.7) Substituting the definition of p and $\tilde{p}$ into the definition of total variation distance gives:

$$d_{TV}\left(p,\tilde{p}\right)=\sum_{\widehat{T}}\left|\boldsymbol{Pr}^{G}\left(\widehat{T}\right)-\mathbb{E}\left[H\sim\mathcal{H}\right]\boldsymbol{Pr}^{H}\left(\widehat{T}\right)\right|.$$

Substituting in the conditions of:

$$\begin{aligned}\boldsymbol{Pr}^{H}\left(\widehat{T}\right)&=\frac{w^{H}\left(T\right)}{\mathcal{T}_{H}},\qquad\text{(by definition of }\boldsymbol{Pr}^{H}(\widehat{T}))\\ w^{H}\left(\widehat{T}\right)&=w^{G}\left(\widehat{T}\right)\cdot\mathbb{P}\left[H'\sim\mathcal{H}\right]\widehat{T}\subseteq {H'}^{-1}\cdot\frac{\mathbb{E}\left[H'\sim\mathcal{H}\right]\mathcal{T}_{H'}}{\mathcal{T}_{G}},\qquad\text{(by given condition)}\end{aligned}$$

Using the fact that

$$\mathbb{E}\left[H\sim\mathcal{H}\right]\mathbf{1}\left(\widehat{T}\subseteq H\right)=\mathbb{P}\left[H'\sim\mathcal{H}\right]\widehat{T}\subseteq H',$$

we can distribute the first term into:

$$d_{TV}\left(p,\tilde{p}\right)=\sum_{\hat{T}}\left|\mathbb{E}\left[H\sim\mathcal{H}\right]\mathbf{1}\left(\widehat{T}\subseteq H\right)\cdot\mathbb{P}\left[H'\sim\mathcal{H}\right]\widehat{T}\subseteq {H'}^{-1}\cdot\boldsymbol{Pr}^{G}\left(\widehat{T}\right)-\boldsymbol{Pr}^{H}\left(\widehat{T}\right)\right|,$$

which by the condition on $w^{H}(\widehat{T})$ simplifies to:

$$d_{TV}\left(p,\tilde{p}\right)=\sum_{\hat{T}}\left|\mathbb{E}\left[H\sim\mathcal{H}\right]\mathbf{1}\left(\widehat{T}\subseteq H\right)\cdot\frac{w^{H}\left(\widehat{T}\right)}{\mathbb{E}\left[H'\sim\mathcal{H}\right]\mathcal{T}_{H'}}-\boldsymbol{Pr}^{H}\left(\widehat{T}\right)\right|.$$

As $\mathbf{1}(\widehat{T} \subseteq H) = 1$ iff $\boldsymbol{Pr}^H(\widehat{T}) > 0$, this further simplifies into

$$d_{TV}(p, \tilde{p}) = \sum_{\hat{T}} \mathbb{P}[H' \sim \mathcal{H}'] \widehat{T} \subseteq H' \left| \mathbb{E}[H \sim \mathcal{H}|_T] \frac{w^H\left(\widehat{T}\right)}{\mathbb{E}[H' \sim \mathcal{H}] \mathcal{T}_{H'}} - \boldsymbol{Pr}^H\left(\widehat{T}\right) \right|,$$

which by triangle inequality gives:

$$d_{TV}(p, \tilde{p}) = \sum_{\hat{T}} \mathbb{P}[H' \sim \mathcal{H}'] \widehat{T} \subseteq H' \cdot \mathbb{E}[H \sim \mathcal{H}|_T] \left| \frac{w^H\left(\widehat{T}\right)}{\mathbb{E}[H' \sim \mathcal{H}] \mathcal{T}_{H'}} - \boldsymbol{Pr}^H\left(\widehat{T}\right) \right|,$$

at which point we can rearrange the summation to obtain:

$$\begin{aligned} d_{TV}(p, \tilde{p}) &\leq \mathbb{E}[H] \sum_{\widehat{T} \subseteq H} \left| \boldsymbol{Pr}^H\left(\widehat{T}\right) - \frac{w^H\left(\widehat{T}\right)}{\mathbb{E}[H'] \mathcal{T}_{H'}} \right| \\ &= \mathbb{E}[H] \sum_{\widehat{T} \subseteq H} w^H\left(\widehat{T}\right) \cdot \left| \frac{1}{\mathcal{T}_H} - \frac{1}{\mathbb{E}[H'] \mathcal{T}_{H'}} \right|. \end{aligned}$$

which by definition of $\mathcal{T}_H$ simplifies to:

$$d_{TV}(p, \tilde{p}) \leq \mathbb{E}[H] \left| 1 - \frac{\mathcal{T}_H}{\mathbb{E}[H'] \mathcal{T}_{H'}} \right|.$$

By the Cauchy-Schwarz inequality, which for distributions can be instantiated as $\mathbb{E}[X] f(X) \leq \sqrt{\mathbb{E}[X] f(X)^2}$ for any random variable X and function $f(X)$, we then get:

$$d_{TV}(p, \tilde{p}) \leq \sqrt{\mathbb{E}[H] \left(1 - \frac{\mathcal{T}_H}{\mathbb{E}[H'] \mathcal{T}_{H'}}\right)^2} = \sqrt{\mathbb{E}[H] \left(\frac{\mathcal{T}_H}{\mathbb{E}[H'] \mathcal{T}_{H'}}\right)^2 - 1} = \sqrt{\delta}.$$

□

2.10.2 Total Variation Distance Bound from Inverse Probability Concentration

We give here our proof of Lemma 2.8.17, that is a more general bound on total variation distance based upon concentration results of the inverse probabilities.

Lemma 2.10.1. *Let X be a random variable such that $X > 0$ over its entire support, and given some $\delta \geq 0$, such that $\mathbb{E}[]\, X = (1 \pm \delta)\mu$ and* **Var** $[]\, X \leq \delta\mu^2$*, then*

$$\mathbb{P}[]\, |X^{-1} - \mu^{-1}| > 4k\sqrt{\delta}\mu^{-1} \leq \frac{1}{k^2}$$

if $1 < k < \delta^{-1/2}/4$

Proof. Chebyshev's inequality gives

$$\mathbb{P}[]\, |X - (1 \pm \delta)\mu| > k\sqrt{\delta}\mu \leq \frac{1}{k^2}.$$

Furthermore, if we assume X such that

$$|X - (1 \pm \delta)\mu| \leq k\sqrt{\delta}\mu$$

which reduces to

$$\left(1 - 2k\sqrt{\delta}\right)\mu \leq X \leq \left(1 + 2k\sqrt{\delta}\right)\mu.$$

Inverting and reversing the inequalities gives

$$\frac{\mu^{-1}}{1 + 2k\sqrt{\delta}} \leq X^{-1} \leq \frac{\mu^{-1}}{1 - 2k\sqrt{\delta}}.$$

Using the fact that $\frac{1}{1+\epsilon} = 1 - \frac{\epsilon}{1+\epsilon} \leq 1 - \epsilon$ for $\epsilon > 0$, and $\frac{1}{1+\epsilon} = 1 + \frac{\epsilon}{1-\epsilon} \leq 1 + 2\epsilon$ for $\epsilon \leq 1/2$, we can then conclude,

$$\left(1 - 4k\sqrt{\delta}\right)\mu^{-1} \leq X^{-1} \leq \left(1 + 4k\sqrt{\delta}\right)\mu^{-1},$$

which implies

$$\mathbb{P}[]\, \left|X^{-1} - \mu^{-1}\right| > 4k\sqrt{\delta}\mu^{-1} \leq \mathbb{P}[]\, |X - (1 \pm \delta)\,\mu| > k\sqrt{\delta}\mu$$

and proves the lemma. □

This bound does not allow us to bound $\mathbb{E}[X]|X-\mu$ because when X close to 0, the value of X^{-1} can be arbitrarily large, while this bound only bounds the probability of such events by $O(\delta^{-1})$. We handle this by treating the case of X small separately, and account for the total probability of such cases via summations over $\mathcal{I}$ and $\widehat{x}$. First we show that once these distributions are truncated to avoid the small X case, its variance is bounded.

Lemma 2.10.2. *Let Y be a random variable such that for parameters $\delta, \mu_Y > 0$ we have $0 < Y \leq 2\mu_Y$ over its entire support, and that $\mathbb{E}[]\, Y^{-1} = (1 \pm \delta)\mu_Y^{-1}$,* **Var** $[]\, Y^{-1} \leq \delta\mu_Y^{-2}$, *then*

$$\mathbb{E}[]\,|Y-\mu_Y| \leq O\left(\sqrt{\delta}\right)\mu_Y.$$

Proof. Since $|Y-\mu_Y| \leq \mu_Y$, we can decompose this expected value into buckets of 2 via:

$$\mathbb{E}[]\,|Y-\mu_Y| \leq \sum_{i=0}^{\log\left(\delta^{-1/2}/4\right)} \mathbb{P}[Y]\,|Y-\mu_Y| \geq 2^i\sqrt{\delta}\mu_Y \cdot \left(2^i\sqrt{\delta}\mu_Y\right),$$

where the last term is from the guarantee of $Y \leq 1$. Lemma 2.10.1 gives that each of the intermediate probability terms is bounded by $O(2^{-2i})$, while the last one is bounded by $\frac{1}{\delta}$, so this gives a total of

$$\mathbb{E}[]\,|Y-\mu| \leq \sum_{i=0}^{\log\left(\delta^{-1/2}\right)} \left(2^i\sqrt{\delta}\mu_Y\right) O\left(2^{-2i}\right) \leq \sqrt{\delta}\mu_Y$$

□

We can now complete the proof via an argument similar to the proof of Lemma 2.4.7 in Section 2.10.1. The only additional step is the definition of BAD_u, which represents the portion of the random variable P_u with high deviation.

Proof of Lemma 2.8.17. For each u, we define a scaling factor corresponding to the proba-

bility that P_u is non-zero:

$$p_{u+} \stackrel{\text{def}}{=} \mathbb{P}\left[p \sim P_u\right] p > 0.$$

By triangle inequality, we have for each P_u

$$\left|1 - \mathbb{E}\left[\right] P_u\right| \leq p_{u+} \cdot \mathbb{E}\left[p \sim P_u \left| p > 0\right.\right] \left|p_{u+}^{-1} - p\right|.$$

We will handle the case where p is close and far from p_{u+}^{-1} separately. This requires defining the portion of P_u with non-zero values, but large variance as

$$BAD_u \stackrel{\text{def}}{=} \left\{p \in \text{supp}\left(P_u\right) : \left|p_{u+}^{-1} - p\right| > \frac{1}{2} p_{u+}^{-1}\right\}.$$

Lemma 2.10.1 gives that for each u,

$$\mathbb{P}\left[p \sim P_u \left| p > 0\right.\right] p \in BAD_u \leq O\left(\sqrt{\delta}\right),$$

which with the outer distribution and factoring the value of p_{u+}^{-1} gives gives:

$$\mathbb{E}\left[u \sim \mathcal{U}\right] p_{u+} \cdot \mathbb{E}\left[p \sim P_u \left| p > 0\right.\right] \mathbf{1}\left(p \in BAD_u\right) \cdot p_{u+}^{-1} \leq O\left(\sqrt{\delta}\right), \tag{2.5}$$

$$\mathbb{E}\left[u \sim \mathcal{U}\right] p_{u+} \cdot \mathbb{E}\left[p \sim P_u \left| p > 0\right.\right] \mathbf{1}\left(p \notin BAD_u\right) \cdot p_{u+}^{-1} \geq 1 - O\left(\sqrt{\delta}\right). \tag{2.6}$$

We then define the 'fixed' distributions $\widetilde{P}_u$ with the same distribution over p as P_u, but whose values are set to p_{u+}^{-1} whenever $p \in BAD_u$. Lemma 2.10.2 then gives:

$$\mathbb{E}\left[p \sim \widetilde{P}_u \left| p > 0\right.\right] \left|p_{u+}^{-1} - p\right| \leq O\left(\sqrt{\delta} p_{u+}^{-1}\right),$$

or taken over the support of $\mathcal{U}$, and written with indicator variables:

$$\mathbb{E}\left[u \sim \mathcal{U}\right] p_{u+} \cdot \mathbb{E}\left[p \sim P_u \left| p > 0\right.\right] \mathbf{1}\left(p \notin BAD_u\right) \cdot \left|p_{u+}^{-1} - p\right| \leq O\left(\sqrt{\delta}\right).$$

Combining this with the lower bound on the mass of p_{u+}^{-1} on the complements of the bad sets from Equation 2.6 via the triangle inequality $p \geq \boldsymbol{p}_{u+}^{-1} - |\boldsymbol{p}_{u+}^{-1} - p|$ gives:

$$\mathbb{E}\left[u \sim \mathcal{U}\right] p_{u+} \cdot \mathbb{E}\left[p \sim P_u \,|p > 0\right] \mathbf{1}\left(p \notin BAD_u\right) \cdot p \geq 1 - O\left(\sqrt{\delta}\right),$$

or upon taking complement again:

$$\mathbb{E}\left[u \sim \mathcal{U}\right] p_{u+} \cdot \mathbb{E}\left[p \sim P_u \,|p > 0\right] \mathbf{1}\left(p \in BAD_u\right) \cdot p \leq O\left(\sqrt{\delta}\right),$$

which together with Equation 2.5 and the non-negativity of p_{u+}^{-1} and p gives

$$\mathbb{E}\left[u \sim \mathcal{U}\right] p_{u+} \cdot \mathbb{E}\left[p \sim P_u \,|p > 0\right] \mathbf{1}\left(p \in BAD_u\right) \cdot \left|p_{u+}^{-1} - p\right| \leq O\left(\sqrt{\delta}\right).$$

Combining these two summations, and invoking the triangle inequality at the start then gives the bound. □

2.11 Deferred Proofs

We now provide detailed proofs of the combinatorial facts about random subsets of edges that are discussed briefly in Section 2.4.

Proof. (of Lemma 2.4.1)

This probability is obtained by dividing the number of subsets of s edges that contain the $n-1$ edges in T, against the number of subsets of s edges from m, which using $\binom{a}{b} = \frac{(a)_b}{(b)_b}$, gives:

$$\binom{m-n+1}{s-n+1} / \binom{m}{s} = \frac{(m-n+1)_{s-n+1}\,(s)_s}{(m)_s\,(s-n+1)_{s-n+1}}, \tag{2.7}$$

and the two terms can be simplified by the rule $(a)_b/(a-k)_{b-k} = (a)_k$.

Furthermore,

$$(a)_b = a^b \left(1 - \frac{1}{a}\right) \cdots \left(1 - \frac{b-1}{a}\right) = a^b \exp\left(\sum_{i=1}^{b-1} \ln\left(1 - \frac{i}{a}\right)\right)$$

We then use the Taylor expansion of $\ln(1-x) = -\sum_{i=1}^{\infty} \frac{x^i}{i}$ to obtain

$$= a^b \exp\left(-\frac{\sum_{i=1}^{b-1} i}{a} - \frac{\sum_{i=1}^{b-1} i^2}{2a^2} - \frac{\sum_{i=1}^{b-1} i^3}{3a^3} - ....\right) = a^b \exp\left(-\frac{b^2}{2a} - O\left(\frac{b^3}{a^2}\right)\right)$$

Substituting into $\frac{(s)_{n-1}}{(m)_{n-1}}$ gives

$$p^{n-1} \exp\left(-\frac{n^2}{2s} + \frac{n^2}{2m} - O\left(\frac{n^3}{s^2}\right) + O\left(\frac{n^3}{m^2}\right)\right) = p^{n-1} \exp\left(-\frac{n^2}{2s} - O\left(\frac{n^3}{s^2}\right)\right)$$

where $\frac{n^2}{2m}$ is absorbed by $O\left(\frac{n^3}{s^2}\right)$ because $m \geq \frac{s^2}{n}$ was assumed.

□

Proof. (Of Lemma 2.4.2)

As before, we have

$$\mathbb{P}\left[H\right] T_1, T_2 \in H = p^{|T_1 \cup T_2|} \exp\left(-\frac{|T_1 \cup T_2|^2}{2s} - O\left(\frac{n^3}{s^2}\right)\right)$$

Invoking the identity:

$$|T_1 \cup T_2| = 2n - 2 - |T_1 \cap T_2|$$

gives

$$\mathbb{P}\left[H\right] T_1, T_2 \in H = p^{2n-2} p^{-k} \exp\left(-\frac{(2n-2-k)^2}{2s} - O\left(\frac{n^3}{s^2}\right)\right).$$

Using the algebraic identity

$$(2n-2-k)^2 \geq 4n^2 + 4nk$$

and dropping the trailing (negative) lower order term gives:

$$\mathbb{P}\left[H\right] T_1, T_2 \in H \leq p^{2n-2} \cdot p^{-k} \exp\left(-\frac{4n^2}{2s} + \frac{4nk}{2s}\right),$$

upon which we can pull out the $\frac{4n^2}{2s}$ term in the exponential to get a term that only depends k. Grouping the p^{-k} term together with the $\exp(\frac{2n}{s})^k$ term, and using the fact that $\exp(t) \leq 1 + 2t$ when $t \leq 0.1$ then gives the result. □

Proof. (of Lemma 2.4.5) We first separate the summation in terms of all possible forests F of size k that any pair of trees could intersect in

$$\sum_{\substack{T_1,T_2 \\ |T_1 \cap T_2|=k}} w\left(T_1\right) \cdot w\left(T_2\right) = \sum_{\substack{F \subseteq E \\ |F|=k}} \sum_{\substack{T_1,T_2 \\ F=T_1 \cap T_2}} w\left(T_1\right) \cdot w\left(T_2\right)$$

We then consider the inner summation, the number of pairs of trees T_1, T_2 with $T_1 \cap T_2 = F$ for some particular set F of size k. This is upper bounded by the square of the number of trees containing F:

$$\sum_{\substack{T_1,T_2 \\ F=T_1 \cap T_2}} w\left(T_1\right) \cdot w\left(T_2\right) \leq \sum_{\substack{T_1,T_2 \\ F \subseteq T_1 \cap T_2}} w\left(T_1\right) \cdot w\left(T_2\right) = \left(\sum_{T:F \subseteq T} w\left(T\right)\right)^2$$

This allow us to directly incorporate the bounds from Lemma 2.4.4, and in turn the assumption of $\boldsymbol{\tau}_e \leq \frac{n}{m}$ to obtain the bound:

$$\sum_{\substack{T_1,T_2 \\ F=T_1 \cap T_2}} w\left(T_1\right) \cdot w\left(T_2\right) \leq \left(\mathcal{T}_G \left(\frac{n}{m}\right)^k\right)^2.$$

Furthermore, the number of possible subsets of F is bounded by $\binom{m}{k}$, which can be bounded even more crudely by $\frac{m^k}{k!}$. Incorporating this then gives:

$$\sum_{\substack{T_1,T_2 \\ |T_1 \cap T_2|=k}} w\left(T_1\right) \cdot w\left(T_2\right) \leq \frac{m^k}{k!} \cdot \left(\mathcal{T}_G \left(\frac{n}{m}\right)^k\right)^2 = \mathcal{T}_G^2 \cdot \frac{1}{k!} \left(\frac{n^2}{m}\right)^k.$$

□

CHAPTER 3

ON THE COMPLEXITY OF NASH EQUILIBRIA IN ANONYMOUS GAMES

This was joint work with Xi Chen and Anthi Orfanou.

3.1 Abstract

We show that the problem of finding an ϵ-approximate Nash equilibrium in an anonymous game with seven pure strategies is complete in PPAD, when the approximation parameter ϵ is exponentially small in the number of players.

3.2 Introduction

The celebrated theorem of Nash [114, 115] states that every game has an equilibrium point. The concept of Nash equilibrium has been tremendously influential in economics and social sciences ever since (e.g., see [116]), and its computation has been one the most well-studied problems in the area of Algorithmic Game Theory. For normal form games with a bounded number of players, much progress has been made during the past decade in understanding both the complexity of Nash equilibrium [117, 118, 119, 120, 121, 122, 123] as well as its efficient approximation [124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135].

In this paper we study a large and important class of *succinct multiplayer* games called *anonymous games* (see [136, 137, 138, 139, 140] for studies of such games in the economics literature). These are special multiplayer games in that the payoff of each player depends only on (1) the pure strategy of the player herself, and (2) the *number* of other players playing each pure strategy, instead of the full pure strategy profile. In such a game, the (expected) payoff of a player is *highly symmetric* over (pure or mixed) strategies of other players. For instance, two players switching their strategies would not affect the payoff

of any other player. A consequence of this very special payoff structure is that $O(\alpha n^{\alpha-1})$ numbers suffice to completely describe the payoff function of a player, when there are α pure strategies shared by n players. Notably this is polynomial in the number of players when α is bounded, and hence the game is succinctly representable. *Throughout the paper, we focus on succinct anonymous games with a bounded number of pure strategies.*

Other well-studied multiplayer games with a succinct representation include graphical, symmetric, and congestion games (for more details see [141]). While graphical and congestion games are both known to be hard to solve [142, 143, 144], there is indeed a polynomial-time algorithm for computing an exact Nash equilibrium in a symmetric game [141]. Because anonymous games generalize symmetric games by allowing player-dependent payoff functions, it is a natural question to ask whether there is an efficient algorithm for finding an (exact or approximate) Nash equilibrium in an anonymous game as well.

Culminating in a sequence of beautiful papers [145, 146, 147, 148, 149] Daskalakis and Papadimitriou obtained a polynomial-time approximation scheme (PTAS) for ϵ-approximate Nash equilibria in anonymous games with a bounded number of strategies (see more discussion on related work in Section 3.2.1). However, the complexity of finding an exact Nash equilibrium in such games remains open, and was conjectured to be hard for PPAD in [148, 149].[1]

In this paper we give an affirmative answer to the conjecture of Daskalakis and Papadimitriou, by showing that it is PPAD-complete to find an ϵ-approximate Nash equilibrium in an anonymous game, when the approximation parameter ϵ is exponentially small in n. To formally state our main result, let (α, c)-ANONYMOUS denote the problem of finding a (2^{-n^c})-approximate Nash equilibrium in an anonymous game with α pure strategies and

[1] When the number of pure strategies is a sufficiently large constant, an anonymous game with rational payoffs may not have any rational equilibrium (e.g., by embedding in it a rational three-player game with no rational equilibrium). But for the case of two strategies, it remains unclear as whether every rational anonymous game has a rational Nash equilibrium, which was posed as an open problem in [149].

payoffs from $[0, 1]$. [2]

Here is our main theorem:

Theorem 3.2.1. *For any $\alpha \geq 7$ and $c > 0$, the problem (α, c)-*ANONYMOUS *is PPAD-complete.*

The greatest challenge to establishing the PPAD-completeness result stated above is posed by the rather complex but also highly symmetric payoff structure of anonymous games. Before discussing our approach and techniques in Section 3.2.3, we first review related work in Section 3.2.1, then define anonymous games formally and introduce some useful notation in Section 3.2.2.

3.2.1 Related Work

Anonymous games have been studied extensively in the economics literature [136, 150, 137, 138, 139, 140, 151], where the game being considered is usually nonatomic and consists of a continuum of players but a finite number of strategies. For the discrete setting, two special families of anonymous games are symmetric games [141, 152] and congestion games [153]. [141] gave a polynomial-time for finding an exact Nash equilibrium in a symmetric game. For congestion games, PLS-completeness of pure equilibria was established in [142, 143, 144][3], and efficient approximation algorithms for various latency functions were obtained in [154, 155, 156].

While an anonymous game does not possess a pure Nash equilibrium in general, it was shown in [145, 157, 149] that when the payoff functions are λ-Lipschitz, there exists an ϵ-approximate pure Nash equilibrium and it can be found in polynomial time, where ϵ has a linear dependency on λ. Furthermore, in [158] Babichenko presented a best-reply dynamic for λ-Lipschitz anonymous games with two strategies which reaches an approximate pure equilibrium in $O(n \log n)$ steps.

[2] Since we are interested in the additive approximation, all payoffs are normalized to take values in $[0, 1]$.

[3] These PLS-hardness results have no implication to the setup of this paper since the number of pure strategies in the congestion games considered there are unbounded.

Regarding our specific point of interest, i.e., (mixed) Nash equilibria in anonymous games with a scaling number of players but a non-scaling number of strategies, there have been a sequence of positive and negative results obtained by Daskalakis and Papadimitriou [145, 147, 146, 148] (summarized in the journal version [149]). We briefly review these results below.

In [145], Daskalakis and Papadimitriou presented a PTAS for finding an ϵ-approximate Nash equilibrium in an anonymous game with two pure strategies, with running time $n^{O(1/\epsilon^2)} \cdot U$, where U denotes the number of bits required to describe the payoffs. The running time was subsequently improved in [146] to $\text{poly}(n) \cdot (1/\epsilon)^{O(1/\epsilon^2)} \cdot U$. The first PTAS in [145] is based on the existence of an ϵ-approximate Nash equilibrium consisting of integer multiples of ϵ^2, while the second PTAS in [146] is based on the existence of an ϵ-approximate Nash equilibrium satisfying the following special property: either at most $O(1/\epsilon^3)$ players play mixed strategies, or all players who mix play the same mixed strategy. Later [147] extended the result of [145], giving the only known PTAS for anonymous games with any bounded number of pure strategies with time $n^{g(\alpha,1/\epsilon)} \cdot U$ for some function g of α, number of pure strategies, and $1/\epsilon$.

All three PTAS obtained in [145, 146, 147] are so-called *oblivious* algorithms [148], i.e., algorithms that enumerate a set of mixed strategy profiles that is independent of the input game as candidates for approximate Nash equilibria (hence, the game is used only to verify if a given mixed strategy profile is an ϵ-approximate Nash equilibrium). In [148], Daskalakis and Papadimitriou showed that any oblivious algorithm for anonymous games must have running time exponential in $1/\epsilon$. In contrast to this negative result, they also presented a *non-oblivious* PTAS for two-strategy anonymous games with running time $\text{poly}(n) \cdot (1/\epsilon)^{O(\log^2(1/\epsilon))} \cdot U$.

3.2.2 Anonymous Games and Polymatrix Games

Before giving a high-level description of our approach and techniques in Section 3.2.3, we first give a formal definition of anonymous games and introduce some useful notation. Consider a multiplayer game with n players $[n] = \{1, \ldots, n\}$ and α pure strategies $[\alpha] = \{1, \ldots, \alpha\}$ with α being a constant. For each pure strategy $b \in [\alpha]$, let $\psi_b(\boldsymbol{t})$ denote the number of b's in a tuple $\boldsymbol{t} \in [\alpha]^{n-1}$, and define $\Psi(\boldsymbol{t}) = (\psi_1(\boldsymbol{t}), \ldots, \psi_\alpha(\boldsymbol{t}))$, which we will refer to as the *histogram* of pure strategies in $\boldsymbol{t}$.

In an anonymous game, the payoff of each player $p \in [n]$ depends only on $\Psi(\boldsymbol{s}_{-p})$ and her own strategy s_p, given a pure strategy profile $\boldsymbol{s} \in [\alpha]^n$. (We follow the convention and use $\boldsymbol{s}_{-p} \in [\alpha]^{n-1}$ to denote the pure strategy profile of the $n-1$ players other than player p in $\boldsymbol{s}$.) Informally, $\Psi(\boldsymbol{s}_{-p})$ can be described as what player p "sees" in the game when $\boldsymbol{s}$ is played.

We now formally define anonymous games.

Definition 3.2.2. An anonymous game $\mathcal{G} = (n, \alpha, \{\textsf{payoff}_p\})$ consists of a set $[n]$ of n players, a set $[\alpha]$ of α pure strategies, and a payoff function $\textsf{payoff}_p : [\alpha] \times K \to \mathbb{R}$ for each player $p \in [n]$, where

$$K = \left\{(k_1, \ldots, k_\alpha) : k_j \in \mathbb{Z}_{\geq 0} \text{ for all } j \text{ and } \textstyle\sum_{j=1}^{\alpha} k_j = n-1\right\}$$

is the set of all histograms of pure strategies played by $n-1$ players. Specifically, when $\boldsymbol{s} \in [\alpha]^n$ is played, the payoff of player p is given by $\textsf{payoff}_p(s_p, \Psi(\boldsymbol{s}_{-p}))$.

As usual, a mixed strategy is a probability distribution $\mathbf{x} = (x_1, \ldots, x_\alpha)$, and a mixed strategy profile $\mathcal{X}$ is an ordered tuple of n mixed strategies $(\mathbf{x}_p : p \in [n])$, one for each player p. Given $\mathcal{X}$, let $u_p(b, \mathcal{X})$ denote the *expected payoff* of p playing $b \in [\alpha]$, which has

the following explicit expression:

$$u_p(b, \mathcal{X}) = \sum_{\mathbf{k} \in K} \mathsf{payoff}_p(b, \mathbf{k}) \cdot \boldsymbol{Pr}_{\mathcal{X}}[p, \mathbf{k}],$$

where $\boldsymbol{Pr}_{\mathcal{X}}[p, \mathbf{k}]$ denotes the probability of player p seeing histogram $\mathbf{k}$ under $\mathcal{X}$:

$$\boldsymbol{Pr}_{\mathcal{X}}[p, \mathbf{k}] = \sum_{\boldsymbol{s}_{-p} \in \Psi^{-1}(\mathbf{k})} \left(\prod_{q \neq p} x_{q, s_q} \right).$$

Note that s_q denotes the pure strategy of player q from a profile $\boldsymbol{s}_{-p} \in \Psi^{-1}(\mathbf{k})$. We also use $u_p(\mathcal{X})$ to denote the expected payoff of player p from playing $\mathbf{x}_p$:

$$u_p(\mathcal{X}) = \sum_{b \in [\alpha]} x_{p,b} \cdot u_p(b, \mathcal{X}).$$

It is worth pointing out that, while $u_p(b, \mathcal{X})$ contains exponentially many terms, it can be computed in polynomial time using dynamic programming [145, 149] when α is a constant. For a detailed presentation of the algorithm for 2-strategy anonymous games, see [149]. This then implies that checking whether a given profile $\mathcal{X}$ is a (approximate) Nash equilibrium is in polynomial time.

Next we define (approximate) Nash equilibria of an anonymous game.

Definition 3.2.3. Given an anonymous game $\mathcal{G} = (n, \alpha, \{\mathsf{payoff}_p\})$, we say a mixed strategy profile $\mathcal{X}$ is a *Nash equilibrium* of $\mathcal{G}$ if $u_p(\mathcal{X}) \geq u_p(b, \mathcal{X})$ for all players $p \in [n]$ and strategies $b \in [\alpha]$.

For $\epsilon \geq 0$, we say $\mathcal{X}$ is an *ϵ-approximate Nash equilibrium* if $u_p(\mathcal{X}) + \epsilon \geq u_p(b, \mathcal{X})$ for all $p \in [n]$ and $b \in [\alpha]$. For $\epsilon \geq 0$, we say $\mathcal{X}$ is an *ϵ-well-supported Nash equilibrium* if $u_p(a, \mathcal{X}) + \epsilon < u_p(b, \mathcal{X})$ implies that $x_{p,a} = 0$, for all $p \in [n]$ and $a, b \in [\alpha]$.

As discussed in Section 3.2.3, the hardness part of Theorem 3.2.1 is proved using a polynomial-time reduction from the problem of finding a well-supported Nash equilibrium

in a *polymatrix game* (e.g. see [159]). For our purposes, such a game (with n players and two strategies each player) can be described by a payoff matrix $\mathbf{A} \in [0,1]^{2n \times 2n}$ with $A_{k,\ell} = 0$ for all $k, \ell \in \{2i-1, 2i\}$ and $i \in [n]$.

Each player $i \in [n]$ has two pure strategies that correspond to rows $2i-1$ and $2i$ of $\mathbf{A}$. Let $\mathbf{A}_j$ denote the jth row of $\mathbf{A}$. Given a vector $\mathbf{y} \in \mathbb{R}^{2n}_{\geq 0}$, where (y_{2i-1}, y_{2i}) is the mixed strategy of player i, expected payoffs of player i for playing rows $2i-1$ and $2i$ are $\mathbf{A}_{2i-1} \cdot \mathbf{y}$ and $\mathbf{A}_{2i} \cdot \mathbf{y}$ respectively.

An *ϵ-well-supported Nash equilibrium* of $\mathbf{A}$ is a vector $\mathbf{y} \in \mathbb{R}^{2n}_{\geq 0}$ such that $y_{2i-1} + y_{2i} = 1$ and

$$\mathbf{A}_{2i-1} \cdot \mathbf{y} > \mathbf{A}_{2i} \cdot \mathbf{y} + \epsilon \Rightarrow y_{2i} = 0 \quad \text{and} \quad \mathbf{A}_{2i} \cdot \mathbf{y} > \mathbf{A}_{2i-1} \cdot \mathbf{y} + \epsilon \Rightarrow y_{2i-1} = 0,$$

for all players $i \in [n]$. We need the following result on such games:

Theorem 3.2.4 ([160])**.** *The problem of computing a $(1/n)$-well-supported Nash equilibrium in a*

polymatrix game is PPAD-complete.

3.2.3 Our Approach and Techniques

A commonly used approach to establishing the PPAD-hardness of approximate equilibria is to design gadget games that can perform certain arithmetic operations on entries of mixed strategies of players (e.g. see [120, 121]). Such gadgets would then yield a reduction from the problem of solving a generalized circuit [120, 121], a problem complete in PPAD. However, we realized that this approach may not work well with anonymous games; we found that it was impossible to design an anonymous game $G_=$ that enforces equality constraints.[4]

[4]For example, we can rule out the existence of an anonymous game $G_=$ with 4 players and 2 pure strategies such that $\mathbf{x}$ is a Nash equilibrium of $G_=$ if and only if $x_1 = x_2 \in [\mu, \nu] \subseteq [0,1]$ and $x_3 = x_4 \in [\mu', \nu'] \subseteq [0,1]$, where we use x_i to denote the probability that player i plays the first pure strategy.

Instead we show the PPAD-hardness of anonymous games via a reduction from the problem of finding a $(1/n)$-well-supported equilibrium in a two-strategy polymatrix game (see Section 3.2.2). Given a $2n \times 2n$ polymatrix game $\mathbf{A}$, our reduction constructs an anonymous game $\mathcal{G}_{\mathbf{A}}$ with n "main" players $\{P_1, \ldots, P_n\}$ (and two auxiliary players). We have each main player P_i simulate in a way a player i in the polymatrix game, as discussed below, such that any ϵ-well-supported Nash equilibrium of $\mathcal{G}_{\mathbf{A}}$ with an exponentially small ϵ can be used to recover a $(1/n)$-well-supported Nash equilibrium of the polymatrix game $\mathbf{A}$ efficiently. We then prove a connection between approximate Nash equilibria and well-supported Nash equilibria of anonymous games to finish the proof of Theorem 3.2.1.

The greatest challenge to establishing such a reduction is posed by the complex but highly structured, symmetric expression of expected payoffs in an anonymous game. As discussed previously in Section 3.2.2, the expected payoff $u_p(b, \mathcal{X})$ of player p is a linear form of probabilities $\boldsymbol{Pr}_{\mathcal{X}}[p, \mathbf{k}]$, each of which is function over mixed strategies of all players other than p. This rather complex function makes it difficult to reason about the set of well-supported Nash equilibria of an anonymous game, not to mention our goal is to embed a polymatrix game in it. To overcome this obstacle, we need to find a special (but hard enough) family of anonymous games with certain payoff structures which allow us to perform a careful analysis and understand their well-supported equilibria. The bigger obstacle for our reduction, however, is to in some sense *remove the anonymity of the players and break the inherent symmetry underlying an anonymous game*.

To see this, a natural approach to obtain a reduction from polymatrix games is to directly encode the $2n$ variables of $\mathbf{y}$ in mixed strategies of the n "main" players $\{P_1, \ldots, P_n\}$. More specifically, let $\{s_1, s_2\}$ denote two special pure strategies of $\mathcal{G}_{\mathbf{A}}$, and we attempt to encode (y_{2i-1}, y_{2i}) in (x_{i,s_1}, x_{i,s_2}), probabilities of P_i playing s_1, s_2, respectively. The reduction would work if expected payoffs of P_i from s_1 and s_2 in $\mathcal{G}_{\mathbf{A}}$ can always match closely expected payoffs of player i from rows $2i-1$ and $2i$ in $\mathbf{A}$, given by two linear forms $\mathbf{A}_{2i-1} \cdot \mathbf{y}$ and $\mathbf{A}_{2i} \cdot \mathbf{y}$ of $\mathbf{y}$. However, it seems difficult, if not impossible, to construct $\mathcal{G}_{\mathbf{A}}$

with this property, since anonymous games are highly symmetric: the expected payoff of P_i is a symmetric function over mixed strategies of all other players. This is not the case for polymatrix games: a linear form such as $\mathbf{A}_{2i} \cdot \mathbf{y}$ in general has different coefficients for different variables, so different players contribute with different weights to the expected payoff of a player (and the problem of finding a well-supported equilibrium in $\mathbf{A}$ clearly becomes trivial if we require that every row of $\mathbf{A}$ has the same entry).

An alternative approach is to encode the $2n$ variables of $\mathbf{y}$ in probabilities $\boldsymbol{Pr}_{\mathcal{X}}[p, \mathbf{k}]$. This may look appealing because expected payoffs $u_p(b, \mathcal{X})$ are linear forms of these probabilities so one can set the coefficients $\mathsf{payoff}_p(b, \mathbf{k})$ to match them easily with those linear forms $\mathbf{A}_j \cdot \mathbf{y}$ that appear in the polymatrix game $\mathbf{A}$. However, the histogram $\mathbf{k}$ seen by a player p (as a vector-valued random variable) is the sum of $n-1$ vector-valued random variables, each distributed according to the mixed strategy of a player other than p. The way these probabilities $\boldsymbol{Pr}_{\mathcal{X}}[p, \mathbf{k}]$ are derived in turn imposes strong restrictions on them,[5] which makes it a difficult task to obtain a correspondence between the $2n$ free variables in $\mathbf{y}$ and the probabilities $\boldsymbol{Pr}_{\mathcal{X}}[p, \mathbf{k}]$.

Our reduction indeed follows the first approach of encoding (y_{2i-1}, y_{2i}) in (x_{i,s_1}, x_{i,s_2}) of player P_i. More exactly, the former is the normalization of the latter into a probability distribution. Now to overcome the difficulty posed by symmetry, we *enforce* the following *"scaling" property* in every well-supported Nash equilibrium $\mathcal{X}$ of $\mathcal{G}_{\mathbf{A}}$: probabilities of P_i playing $\{s_1, s_2\}$ satisfy

$$x_{i,s_1} + x_{i,s_2} \approx 1/N^i, \tag{3.1}$$

where N is exponentially large in n. This property is established by designing an anonymous game called *generalized radix game* $\mathcal{G}^*_{n,N}$, and then using it as the base game in the construction of $\mathcal{G}_{\mathbf{A}}$. We show that (3.1) holds approximately for every anonymous game that is payoff-wise *close* to $\mathcal{G}^*_{n,N}$. In particular, (3.1) holds for any well-supported equilibrium of

[5]For example, as it is pointed out in [145, 147] for anonymous games with two strategies, players can always be partitioned into two sets such that the probabilities $\boldsymbol{Pr}_{\mathcal{X}}[p, \mathbf{k}]$ over $\mathbf{k}$ must follow approximately a Poisson and a discretized Normal distribution on each set respectively.

$\mathcal{G}_{\mathbf{A}}$, as long as we make sure $\mathcal{G}_{\mathbf{A}}$ is close to $\mathcal{G}^*_{n,N}$. The "scaling" property plays a crucial role in our reduction because, as the base game for $\mathcal{G}_{\mathbf{A}}$, it helps us reason about well-supported Nash equilibria of $\mathcal{G}_{\mathbf{A}}$; it also removes anonymity of the n "main" players P_i (since they must play the two special pure strategies $\{s_1, s_2\}$ with probabilities of different scales) and overcome the symmetry barrier.

Equipped with the "scaling" property (3.1), we prove a key technical lemma called the *estimation lemma*. It shows that one can compute efficiently coefficients of a linear form over probabilities of histograms $\boldsymbol{Pr}_{\mathcal{X}}[P_i, \mathbf{k}]$ seen by player P_i, which guarantees to approximate additively x_{j,s_1} (or x_{j,s_2}) i.e. probability of another player P_j plays s_1 (or s_2), whenever the profile $\mathcal{X}$ satisfies the "scaling" property (this holds when $\mathcal{G}_{\mathbf{A}}$ is close to $\mathcal{G}^*_{n,N}$ and $\mathcal{X}$ is a well-supported equilibrium of $\mathcal{G}_{\mathbf{A}}$). As

$$(y_{2j-1}, y_{2j}) \approx N^j(x_{j,s_1}, x_{j,s_2})$$

given (3.1), these linear forms for x_{j,s_1}, x_{j,s_2} can be combined to derive a linear form of $\boldsymbol{Pr}_{\mathcal{X}}[P_i, \mathbf{k}]$ to approximate additively any linear form of $\mathbf{y}$, particularly $\mathbf{A}_{2i-1} \cdot \mathbf{y}$ or $\mathbf{A}_{2i} \cdot \mathbf{y}$ that appear as expected payoffs of player i in the polymatrix game $\mathbf{A}$. The proof of the estimation lemma is the technically most involved part of the paper. We indeed derive explicit expressions for coefficients of the desired linear form where substantial cancellations yield an additive approximation of x_{j,s_1} or x_{j,s_2}.

Finally we combine all ingredients highlighted above to construct an anonymous game $\mathcal{G}_{\mathbf{A}}$ from polymatrix game $\mathbf{A}$. This is done by first using the estimation lemma to compute, for each main P_i coefficients of linear forms of probabilities $\boldsymbol{Pr}_{\mathcal{X}}[P_i, \mathbf{k}]$ seen by P_i that yield additive approximations of x_{j,s_1} and x_{j,s_2}. We then perturb payoff functions of players P_i in the generalized radix game $\mathcal{G}^*_{n,N}$ using these coefficients so that 1) the resulting game $\mathcal{G}_{\mathbf{A}}$ is close to $\mathcal{G}^*_{n,N}$ and thus, any well-supported equilibrium $\mathcal{X}$ of $\mathcal{G}_{\mathbf{A}}$ automatically satisfies the "scaling" property; 2) expected payoffs of P_i playing s_1, s_2 in a well-supported equilibrium

$\mathcal{X}$ of $\mathcal{G}_{\mathbf{A}}$ match additively expected payoffs of player i playing rows $2i-1, 2i$ in $\mathbf{A}$, given $\mathbf{y}$ derived from $\mathcal{X}$ by normalizing (x_{j,s_1}, x_{j,s_2}) for each j. The correctness of the reduction, i.e., $\mathbf{y}$ is a $(1/n)$-well-supported equilibrium of $\mathbf{A}$ whenever $\mathcal{X}$ is an ϵ-approximate equilibrium of $\mathcal{G}_{\mathbf{A}}$ with an exponentially small ϵ, follows from these properties of $\mathcal{G}_{\mathbf{A}}$.

3.2.4 Organization

In Section 2, we define the radix game, and show that it has a unique Nash equilibrium as a warm-up. We also use it to define the generalized radix game which serves as the base of our reduction. In section 3, we characterize well-supported Nash equilibria of anonymous games that are close to the generalized radix game (i.e., those that can be obtained by adding small perturbations to payoffs of the generalized radix game). In section 4, we prove the PPAD-hardness part of the main theorem. Our reduction relies on a crucial technical lemma, called the estimation lemma, which we prove in Section 5. We prove the membership in Section 6, and conclude with open problems in Section 7.

3.3 Warm-up: Radix Game

In this section, we first define a $(n+2)$-player anonymous game $\mathcal{G}_{n,N}$, called the *radix game*. As a warmup for the next section, we show that it has a unique Nash equilibrium. We then use the radix game to define the *generalized radix game* $\mathcal{G}^*_{n,N}$, by making a duplicate of a pure strategy in $\mathcal{G}_{n,N}$. The latter will serve as the base game for our polynomial-time reduction from polymatrix games.

3.3.1 Radix Game

The radix game $\mathcal{G}_{n,N}$ to be defined has a unique Nash equilibrium of a specific form: given $N \geq 2$ as an integer parameter of the game, each of the n "main" players mixes over the first two strategies with probabilities $1/N^i$ and $1-1/N^i$, respectively, for each $i \in [n]$, in the unique Nash equilibrium. The remaining two "special" players are created to achieve

the aforementioned property.

Game 1 (Radix Game $\mathcal{G}_{n,N}$)**.** Let $n \geq 1$ and $N \geq 2$ denote two integer parameters. Let $\delta = 1/N$.

Let $\mathcal{G}_{n,N}$ denote the following anonymous game with $n+2$ players $\{P_1, \ldots, P_n, Q, R\}$ and 6 pure strategies $\{s, t, q_1, q_2, r_1, r_2\}$. We refer to $\{P_1, \ldots, P_n\}$ as the *main* players. Each main player P_i is only interested in strategies s and t *(*e.g., by setting her payoff of playing any other four actions to be -1 no matter what other players play*)*. Player Q is only interested in strategies $\{q_1, q_2\}$, and player R is only interested in strategies $\{r_1, r_2\}$.

Next we define the payoff function of each player. When describing the payoff of a player below we always use $\mathbf{k} = (k_s, k_t, k_{q_1}, k_{q_2}, k_{r_1}, k_{r_2})$ to denote the histogram of strategies this player sees.

1. For each $i \in [n]$, the payoff of player P_i when she plays s only depends on k_s:

$$\textsf{payoff}_{P_i}(s, \mathbf{k}) = \begin{cases} \delta^i + \prod_{j \in [n]} \delta^j & \text{if } k_s = n-1 \\ \prod_{j \in [n]} \delta^j & \text{otherwise.} \end{cases}$$

   The payoff of player P_i when she plays t only depends on k_{r_1}:

$$\textsf{payoff}_{P_i}(t, \mathbf{k}) = \begin{cases} 2 & \text{if } k_{r_1} = 1 \\ 0 & \text{otherwise.} \end{cases}$$

2. The payoff of player Q when she plays q_1 or q_2 is given by

$$\textsf{payoff}_{Q}(q_1, \mathbf{k}) = \begin{cases} 1 & \text{if } k_s = n \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \textsf{payoff}_{Q}(q_2, \mathbf{k}) = \begin{cases} 1 & \text{if } k_{r_1} = 1 \\ 0 & \text{otherwise.} \end{cases}$$

3. The payoff of player R when she plays r_1 or r_2 is given by

$$\textit{payoff}_R(r_1, \mathbf{k}) = \begin{cases} 1 & \text{if } k_{q_1} = 1 \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \textit{payoff}_R(r_2, \mathbf{k}) = \begin{cases} 1 & \text{if } k_{q_2} = 1 \\ 0 & \text{otherwise.} \end{cases}$$

This finishes the definition of the radix game $\mathcal{G}_{n,N}$.

Fact 3.3.1. *$\mathcal{G}_{n,N}$ is an anonymous game with payoff functions taking values from $[-1, 2]$.*

Since the main players P_i are only interested in $\{s, t\}$, Q is only interested in $\{q_1, q_2\}$, and R is only interested in $\{r_1, r_2\}$, each Nash equilibrium $\mathcal{X}$ of $\mathcal{G}_{n,N}$ can be fully specified by a $(n+2)$-tuple $\mathcal{X} = (x_1, \ldots, x_n, y, z) \in [0, 1]^{n+2}$, where x_i denotes the probability of P_i playing strategy s for each $i \in [n]$, y denotes the probability of Q playing q_1, and z denotes the probability of R playing r_1.

Given $\mathcal{X} = (x_1, \ldots, x_n, y, z)$ we calculate the expected payoff of each player as follows (we skip $\mathcal{X}$ in the expected payoffs $u_p(b, \mathcal{X})$, when $\mathcal{X}$ is clear from the context, and we use u_i to denote the expected payoff of P_i instead of u_{P_i} for convenience):

Fact 3.3.2. *Given $\mathcal{X} = (x_1, \ldots, x_n, y, z)$, the expected payoff of player P_i for playing s is*

$$u_i(s) = \delta^i \cdot \boldsymbol{Pr} k_s = n - 1 + \prod_{j \in [n]} \delta^j = \delta^i \prod_{j \neq i \in [n]} x_j + \prod_{j \in [n]} \delta^j.$$

The expected payoff of P_i for playing t is $u_i(t) = 2z$.

The expected payoff of player Q for playing q_1 is

$$u_Q(q_1) = \boldsymbol{Pr} k_s = n = \prod_{i \in [n]} x_i.$$

The expected payoff of Q for playing q_2 is $u_Q(q_2) = z$.

The expected payoff of R for playing r_1 is $u_R(r_1) = y$ and that for r_2 is $u_R(r_2) = 1 - y$.

We show that $x_i = \delta^i$ in a Nash equilibrium $\mathcal{X}$ of $\mathcal{G}_{n,N}$. We start with the following lemma.

Lemma 3.3.3. *In a Nash equilibrium $\mathcal{X} = (x_1, \ldots, x_n, y, z)$ of $\mathcal{G}_{n,N}$, we have that $z = \prod_{i\in[n]} x_i$.*

Proof. Assume for contradiction that $z > \prod_i x_i$. As $u_Q(q_2) > u_Q(q_1)$ and $\mathcal{X}$ is a Nash equilibrium, player Q never plays q_1 and thus, $y = 0$. This in turn implies $u_R(r_2) = 1 > 0 = u_R(r_1)$ and $z = 0$, which contradicts with the assumption that $z > \prod_i x_i \geq 0$.

Next, assume for contradiction that $z < \prod_i x_i$, giving us that $u_Q(q_2) < u_Q(q_1)$. Player Q never plays q_2 and $y = 1$. This implies that $u_R(r_1) > u_R(r_2)$ and thus $z = 1$, which contradicts with the assumption that $z < \prod_i x_i \leq 1$ (as $x_i \in [0, 1]$). This finishes the proof of the lemma. □

We now show that the radix game $\mathcal{G}_{n,N}$ has a unique Nash equilibrium $\mathcal{X}$ with $x_i = \delta^i$.

Lemma 3.3.4. *In a Nash equilibrium $\mathcal{X} = (x_1, \ldots, x_n, y, z)$ of $\mathcal{G}_{n,N}$, we have $x_i = \delta^i$ for all $i \in [n]$.*

Proof. First we show that $\prod_{i\in[n]} x_i = \prod_{i\in[n]} \delta^i$. Consider for contradiction the following two cases:

Case 1: $\prod_{i\in[n]} x_i < \prod_{i\in[n]} \delta^i$. Then there is an $i \in [n]$ such that $x_i < \delta^i$. For P_i, we have

$$u_i(s) = \delta^i \prod_{j\neq i} x_j + \prod_{j\in[n]} \delta^j > \prod_{j\in[n]} x_j + \prod_{j\in[n]} x_j = 2 \prod_{j\in[n]} x_j = 2z = u_i(t). \quad (3.2)$$

This implies that $x_i = 1$, contradicting with the assumption that $x_i < \delta^i < 1$ as $N \geq 2$.

Case 2: $\prod_{i\in[n]} x_i > \prod_{i\in[n]} \delta^i$. Then there is an $i \in [n]$ such that $x_i > \delta^i$. For P_i, we

have

$$u_i(s) = \delta^i \prod_{j \neq i} x_j + \prod_{j \in [n]} \delta^j < \prod_{j \in [n]} x_j + \prod_{j \in [n]} x_j = 2 \prod_{j \in [n]} x_j = 2z = u_i(t). \quad (3.3)$$

This implies that $x_i = 0$, contradicting with the assumption that $x_i > \delta^i > 0$.

As a result, we must have $\prod_i x_i = \prod_i \delta^i$, which also implies that $x_i > 0$ for all $i \in [n]$.

Now we show that $x_i = \delta^i$ for all i. Assume for contradiction that $x_i \neq \delta^i$ for some $i \in [n]$.

Case 1: $x_i < \delta^i$. Then the same strict inequality (3.2) holds for P_i, which implies that $x_i = 1$, contradicting with the assumption that $x_i < \delta^i < 1$ as $N \geq 2$.

Case 2: $x_i > \delta^i$. Then the same strict inequality (3.3) holds for P_i, which implies that $x_i = 0$, contradicting with the assumption that $x_i > \delta^i > 0$.

This finishes the proof of the lemma. □

Notice that Lemma 3.3.3 and 3.3.4 together imply that $\mathcal{G}_{n,N}$ has a unique Nash equilibrium because of Lemma 3.3.3 as well as the fact that $0 < z < 1$ implies $u_R(r_1) = y = 1 - y = u_R(r_2)$ and thus $y = 1/2$.

3.3.2 Generalized Radix Game

We use $\mathcal{G}_{n,N}$ to define an anonymous game $\mathcal{G}^*_{n,N}$, called the *generalized radix game*, with the same set of $n + 2$ players $\{P_1, \ldots, P_n, Q, R\}$ but seven strategies $\{s_1, s_2, t, q_1, q_2, r_1, r_2\}$. To this end, we replace strategy s in $\mathcal{G}_{n,N}$ with two of its duplicate strategies s_1, s_2 in $\mathcal{G}^*_{n,N}$ and make sure that the players in $\mathcal{G}^*_{n,N}$ treat both s_1 and s_2 the same as the old strategy s, and have their payoff functions derived from those of players in $\mathcal{G}_{n,N}$ in this fashion. We will show in the next section that in any Nash equilibrium of $\mathcal{G}^*_{n,N}$, player P_i must have probability exactly δ^i distributed among s_1, s_2.

For readers who are familiar with previous PPAD-hardness results of Nash equilibria in normal form games [120, 121], this is the same trick used to derive the game *generalized matching pennies* from *matching pennies*. We define $\mathcal{G}^*_{n,N}$ formally as follows.

Game 2 (Generalized Radix Game $\mathcal{G}^*_{n,N}$)**.** Let $n \geq 1$ and $N \geq 2$ be two parameters. Let $\delta = 1/N$. We use $\mathcal{G}^*_{n,N}$ to denote an anonymous game with the same $n+2$ players $\{P_1, \ldots, P_n, Q, R\}$ as $\mathcal{G}_{n,N}$ but now 7 pure strategies $\{s_1, s_2, t, q_1, q_2, r_1, r_2\}$. The payoff function $\textsf{payoff}^*_T$ of a player T in $\mathcal{G}^*_{n.N}$ is defined using $\textsf{payoff}_T$ of the same player T in $\mathcal{G}_{n,N}$ as follows:

$$\textsf{payoff}^*_T\Big(b, \big(k_{s_1}, k_{s_2}, k_t, k_{q_1}, k_{q_2}, k_{r_1}, k_{r_2}\big)\Big) = \textsf{payoff}_T\Big(\phi(b), \big(k_{s_1}+k_{s_2}, k_t, k_{q_1}, k_{q_2}, k_{r_1}, k_{r_2}\big)\Big),$$

where $\phi(s_1) = \phi(s_2) = s$ and $\phi(b) = b$ for every other pure strategy.

Since the payoff of player P_i is always -1 when playing q_1, q_2, r_1 or r_2, she is only interested in s_1, s_2 and t. Similarly Q is only interested in q_1, q_2 and R is only interested in r_1, r_2. As a result, a Nash equilibrium $\mathcal{X}$ of $\mathcal{G}^*_{n,N}$ can be fully specified by $2n+2$ numbers $(x_{i,1}, x_{i,2}, y, z : i \in [n])$, where $x_{i,1}$ (or $x_{i,2}$) denotes the probability of P_i playing strategy s_1 (or strategy s_2, respectively), so the probability of P_i playing t is $1 - x_{i,1} - x_{i,2}$. We also let $x_i = x_{i,1} + x_{i,2}$ for each $i \in [n]$.

Given the definition of $\mathcal{G}^*_{n,N}$ from $\mathcal{G}_{n,N}$, Lemma 3.3.4 suggests $x_i = x_{i,1} + x_{i,2} = \delta^i$, for all $i \in [n]$, in every Nash equilibrium $\mathcal{X}$ of $\mathcal{G}^*_{n,N}$. This indeed follows from the main lemma of the next section concerning ϵ-well-supported Nash equilibria of not only the generalized radix game $\mathcal{G}^*_{n,N}$ itself, but also anonymous games obtained by perturbing payoff functions of $\mathcal{G}^*_{n,N}$.

3.4 Generalized Radix Game after Perturbation

In this section, we analyze ϵ-well-supported Nash equilibria of anonymous games obtained by perturbing payoff functions of the generalized radix game $\mathcal{G}^*_{n,N}$. Recall that $n \geq 1$ and

$N \geq 2$, and we use payoff^*_T to denote the payoff function of a player T in $\mathcal{G}^*_{n,N}$. Given $x, y \in \mathbb{R}$ and $\xi \geq 0$, we write $x = y \pm \xi$ to denote $|x - y| \leq \xi$. We first define anonymous games that are close to $\mathcal{G}^*_{n,N}$.

Definition 3.4.1. For $\xi \geq 0$, we say an anonymous game $\mathcal{G}$ is ξ-*close* to $\mathcal{G}^*_{n,N}$ if

1. $\mathcal{G}$ has the same set $\{P_1, \ldots, P_n, Q, R\}$ of players and same set of 7 strategies as $\mathcal{G}^*_{n,N}$.
2. For each player $T \in \{P_1, \ldots, P_n, Q, R\}$, her payoff function payoff_T in $\mathcal{G}$ satisfies

$$\mathit{payoff}_T(b, \mathbf{k}) = \mathit{payoff}^*_T(b, \mathbf{k}) \pm \xi,$$

for all $b \in \{s_1, s_2, t, q_1, q_2, r_1, r_2\}$ and all histograms $\mathbf{k}$ of strategies played by $n + 1$ players.

To characterize ϵ-well-supported Nash equilibria of a game $\mathcal{G}$ ξ-close to $\mathcal{G}^*_{n,N}$ we first show that when ϵ, ξ are small enough, each player in $\mathcal{G}$ remains only interested in a subset of strategies, i.e., $\{s_1, s_2, t\}$ for P_i, $\{q_1, q_2\}$ for Q, and $\{r_1, r_2\}$ for R, in any ϵ-well-supported Nash equilibrium of $\mathcal{G}$.

Lemma 3.4.2. *Let $\mathcal{G}$ be an anonymous game ξ-close to $\mathcal{G}^*_{n,N}$ for some $\xi \geq 0$. When $2\xi + \epsilon < 1$, every ϵ-well-supported Nash equilibrium of $\mathcal{G}$ satisfies: player P_i only plays $\{s_1, s_2, t\}$; player Q only plays $\{q_1, q_2\}$; player R only plays $\{r_1, r_2\}$.*

Proof. We only prove (1) since the proof of (2) and (3) is similar.

Given an ϵ-well-supported Nash equilibrium $\mathcal{X}$, as the payoff of P_i when playing $b \notin \{s_1, s_2, t\}$ is always -1 in $\mathcal{G}^*_{n,N}$, her expected payoff when playing b in $\mathcal{G}$ is at most $-1 + \xi$; as the payoff of P_i when playing $b \in \{s_1, s_2, t\}$ is always nonnegative in $\mathcal{G}^*_{n,N}$, her expected payoff in $\mathcal{G}$ is at least $-\xi$. It follows from $2\xi + \epsilon < 1$ and the assumption of $\mathcal{X}$ being an ϵ-well-supported equilibrium that P_i only plays strategies in $\{s_1, s_2, t\}$ with positive probability. □

It follows from Lemma 3.4.2 that an ϵ-well-supported Nash equilibrium of $\mathcal{G}$ can be fully described by a tuple of $2n+2$ numbers $(x_{i,1}, x_{i,2}, y, z : i \in [n])$, when ξ, ϵ satisfy $2\xi + \epsilon < 1$: $x_{i,1}$ denotes the probability of P_i playing s_1, $x_{i,2}$ denotes the probability of P_i playing s_2, y denotes the probability of Q playing q_1, and z denotes the probability of R playing r_1.

Recall that $\delta = 1/N \leq 1/2$. Let $\kappa = \prod_{i\in[n]} \delta^i$. We prove the main lemma of this section.

Lemma 3.4.3. *Let $\mathcal{G}$ denote an anonymous game that is ξ-close to $\mathcal{G}^*_{n,N}$. Suppose that $\xi, \epsilon \geq 0$ satisfy*

$$\tau = \frac{36\xi + 18\epsilon}{\kappa} \leq 1/2. \tag{3.4}$$

Then every ϵ-well-supported Nash equilibrium of $\mathcal{G}$ satisfies $x_{i,1} + x_{i,2} = \delta^i \pm \tau\delta^i$ for all $i \in [n]$.

Proof. Let $\mathcal{X} = (x_{i,1}, x_{i,2}, y, z : i \in [n])$ be an ϵ-well-supported Nash equilibrium of $\mathcal{G}$. For each $i \in [n]$ we let $x_i = x_{i,1} + x_{i,2}$. Since $\mathcal{G}$ is ξ-close to $\mathcal{G}^*_{n,N}$, we have the following estimates:

1. The expected payoff of P_i for playing strategy s_1 or s_2 is

$$u_i(s_1),\, u_i(s_2) = \left(\delta^i \cdot \boldsymbol{Pr}\big[k_{s_1} + k_{s_2} = n-1\big] + \prod_{j\in[n]} \delta^j\right) \pm \xi$$
$$= \left(\delta^i \prod_{j\neq i} x_j + \kappa\right) \pm \xi,$$

   where we write k_{s_1}, k_{s_2} to denote the numbers of players that play s_1, s_2 respectively, as seen by player P_i (same below). The expected payoff of P_i for playing t is $u_i(t) = 2z \pm \xi$.

2. The expected payoff of Q for playing q_1 is

$$u_Q(q_1) = \boldsymbol{Pr}\big[k_{s_1} + k_{s_2} = n\big] \pm \xi = \textstyle\prod_{j\in[n]} x_j \pm \xi.$$

The expected payoff of Q for playing q_2 is $u_Q(q_2) = z \pm \xi$.

3. The expected payoff of R for playing r_1 is $u_R(r_1) = y \pm \xi$ and for r_2 is $u_R(r_2) = (1-y) \pm \xi$.

To rest of the proof follows those of Lemma 3.3.3 and Lemma 3.3.4. First we show that z must satisfy

$$z = \prod_{j\in[n]} x_j \pm (2\xi + \epsilon). \tag{3.5}$$

The proof is the same as that of Lemma 3.3.3, using the assumption that $\mathcal{X}$ is ϵ-well-supported.

Given (3.5), next we show that the x_i's satisfy

$$\prod_{i\in[n]} x_i = \prod_{i\in[n]} \delta^i \pm (6\xi + 3\epsilon) = \kappa \pm (6\xi + 3\epsilon). \tag{3.6}$$

To this end we follow the proof of the first part of Lemma 3.3.4 and consider the following two cases:

Case 1: $\prod_{i\in[n]} x_i < \kappa - (6\xi + 3\epsilon)$. Then there exists an $i \in [n]$ such that $x_i < \delta^i$. For P_i:

$$u_i(s_1) \geq \delta^i \prod_{j\neq i} x_j + \kappa - \xi > 2\prod_{j\in[n]} x_j + 5\xi + 3\epsilon \;\text{ and }\; u_i(t) \leq 2z + \xi \leq 2\prod_{j\in[n]} x_j + 5\xi + 2\epsilon.$$

This implies that P_i does not play t in $\mathcal{X}$, an ϵ-well-supported Nash equilibrium of $\mathcal{G}$, and thus, $x_i = x_{i,1} + x_{i,2} = 1$, contradicting with $x_i < \delta^i < 1$ as $N \geq 2$.

Case 2: $\prod_{i\in[n]} x_i > \kappa + (6\xi + 3\epsilon)$. Then there exists an $i \in [n]$ such that $x_i > \delta^i$. For

P_i:

$$u_i(s_1), u_i(s_2) \leq \delta^i \prod_{j \neq i} x_j + \kappa + \xi < 2 \prod_{j \in [n]} x_j - 5\xi - 3\epsilon \text{ and } u_i(t) \geq 2 \prod_{j \in [n]} x_j - 5\xi - 2\epsilon.$$

This implies that P_i plays neither s_1 nor s_2 and thus, we have $x_{i,1} = x_{i,2} = 0$ and $x_i = 0$ as well, contradicting with $x_i > \delta^i > 0$.

By (3.5) and (3.6), $z = \kappa \pm (8\xi + 4\epsilon)$. (3.6) also implies that $x_i > 0$ since $\kappa > 0$ and $\kappa \geq 72\xi + 36\epsilon$ by (3.4).

Finally, assume for contradiction that either $x_i < (1-\tau)\delta^i$ or $x_i > (1+\tau)\delta^i$ for some $i \in [n]$.

Case 1: $x_i < (1-\tau)\delta^i$. Then using $\tau \leq 1/2$ and $1 \leq 1/(1-\tau) \leq 2$, we have

$$\begin{aligned} u_i(s_1) - u_i(t) &\geq \delta^i \prod_{j \neq i} x_j + \kappa - 2z - 2\xi \\ &> \frac{\kappa - 6\xi - 3\epsilon}{1-\tau} + \kappa - 2z - 2\xi \geq \tau\kappa - 30\xi - 14\epsilon. \end{aligned}$$

Plugging in the definition of τ in (3.4), we have $u_i(s_1) - u_i(t) > \epsilon$ and thus, $x_i = 1$, which contradicts with the assumption that $x_i < (1-\tau)\delta^i < 1$.

Case 2: $x_i > (1+\tau)\delta^i$. Then using $\tau \leq 1/2$ and $2/3 \leq 1/(1+\tau) \leq 1$, we have

$$\begin{aligned} u_i(s_1) - u_i(t) &\leq \delta^i \prod_{j \neq i} x_j + \kappa - 2z + 2\xi \\ &< \frac{\kappa + 6\xi + 3\epsilon}{1+\tau} + \kappa - 2z + 2\xi \leq -\frac{2\tau\kappa}{3} + 24\xi + 11\epsilon. \end{aligned}$$

The same inequality holds for $u_i(s_2) - u_i(t)$. Plugging in (3.4), we have $u_i(s_1) - u_i(t) < -\epsilon$ as well as $u_i(s_2) - u_i(t) < -\epsilon$. This in turn implies that $x_{i,1} = x_{i,2} = 0$ and thus, $x_i = 0$, which contradicts with the assumption that $x_i > (1+\tau)\delta^i > 0$.

This finishes the proof of the lemma. □

3.5 Reduction from Polymatrix Games to Anonymous Games

In this section we prove the hardness part of Theorem 3.2.1. For this purpose we present a polynomial time reduction from the problem of finding a $1/n$-well-supported Nash equilibrium in a polymatrix game to the problem of finding an ϵ-well-supported Nash equilibrium in an anonymous game with 7 strategies, for some exponentially small ϵ. We first give some intuition behind this quite involved reduction in Section 3.5.1. Details of the reduction and the proof of its correctness are then presented in Section 3.5.2 and 3.5.3, respectively, with a key technical lemma proved in Section 3.6. We finish the proof of the hardness part in Section 3.5.4 by showing that any approximate Nash equilibrium of an anonymous game can be converted into a well-supported equilibrium efficiently (since Theorem 3.2.1 is concerned with approximate Nash equilibria).

3.5.1 Overview of the Reduction

Given as input a polymatrix game specified by a matrix $\mathbf{A} \in [0,1]^{2n \times 2n}$, our goal is to construct in polynomial time an anonymous game $\mathcal{G}_{\mathbf{A}}$, and show that every ϵ-well-supported Nash equilibrium of $\mathcal{G}_{\mathbf{A}}$, where $\epsilon = 1/2^{n^6}$, can be used to recover a $(1/n)$-well-supported equilibrium of $\mathbf{A}$ in polynomial time. Note that this is not exactly the PPAD-hardness result as claimed in Theorem 3.2.1 but we will fill in the gap in Section 3.5.4 with some standard arguments.

Given $\mathbf{A}$, we construct $\mathcal{G}_{\mathbf{A}}$ by perturbing payoff functions of the Generalized Radix game $\mathcal{G}^*_{n,N}$ with $N = 2^n$, so that $\mathcal{G}_{\mathbf{A}}$ is ξ-close to $\mathcal{G}^*_{n,N}$ for some exponentially small $\xi > 0$ to be specified later. (Thus, $\mathcal{G}_{\mathbf{A}}$ has the same set of $n+2$ players $\{P_1, \ldots, P_n, Q, R\}$ as well as the same set of 7 strategies $\{s_1, s_2, t, q_1, q_2, r_1, r_2\}$ as $\mathcal{G}^*_{n,N}$.) By Lemma 3.4.2 and Lemma 3.4.3 we know that every ϵ-well-supported equilibrium of $\mathcal{G}_{\mathbf{A}}$ can be fully described

by a tuple $\mathcal{X} = (x_{i,1}, x_{i,2}, y, z : i \in [n])$ that satisfies

$$x_{i,1} + x_{i,2} \approx \delta^i \tag{3.7}$$

for each $i \in [n]$, where $\delta = 1/N = 1/2^n$.

Our construction of $\mathcal{G}_{\mathbf{A}}$ has player P_ℓ simulate row $2\ell - 1$ and 2ℓ of the polymatrix game $\mathbf{A}$ for each $\ell \in [n]$. The goal is to show at the end that, after normalizing $(x_{\ell,1}, x_{\ell,2})$, i.e., probabilities of P_ℓ playing s_1, s_2 in an ϵ-well-supported equilibrium $\mathcal{X}$ of $\mathcal{G}_{\mathbf{A}}$, into a distribution $(y_{2\ell-1}, y_{2\ell})$:

$$y_{2\ell-1} = \frac{x_{\ell,1}}{x_{\ell,1} + x_{\ell,2}} \quad \text{and} \quad y_{2\ell} = \frac{x_{\ell,2}}{x_{\ell,1} + x_{\ell,2}}, \tag{3.8}$$

we get a $(1/n)$-well-supported Nash equilibrium $\mathbf{y} = (y_1, \ldots, y_{2n})$ of $\mathbf{A}$. By (3.7) we have

$$y_{2\ell-1} \approx N^\ell \cdot x_{\ell,1} \quad \text{and} \quad y_{2\ell} \approx N^\ell \cdot x_{\ell,2}.$$

For player P_ℓ to simulate row $2\ell - 1$ and 2ℓ of the polymatrix game $\mathbf{A}$, we perturb the original payoff function payoff^*_ℓ of P_ℓ in $\mathcal{G}^*_{n,N}$ in a way such that the following two linear forms of $\mathbf{y}$:

$$\mathbf{A}_{2\ell-1} \cdot \mathbf{y} = \sum_{j \notin \{2\ell-1, 2\ell\}} A_{2\ell-1,j} \cdot y_j \quad \text{and} \quad \mathbf{A}_{2\ell} \cdot \mathbf{y} = \sum_{j \notin \{2\ell-1, 2\ell\}} A_{2\ell,j} \cdot y_j$$

appear as additive terms in the expected payoffs $u_\ell(s_1, \mathcal{X})$ and $u_\ell(s_2, \mathcal{X})$ of P_ℓ obtained from s_1, s_2, respectively. Let $u^*_\ell(\sigma, \mathcal{X})$ denote the expected payoff of player P_ℓ in the original generalized radix game $\mathcal{G}^*_{n,N}$ for strategies $\sigma \in \{s_1, s_2\}$. Then more specifically, we would like to perturb carefully the payoff functions of $\mathcal{G}^*_{n,N}$ such that for every $\ell \in [n]$, the expected payoffs of player P_ℓ in an ϵ-well-supported Nash equilibrium $\mathcal{X}$ of $\mathcal{G}_{\mathbf{A}}$ satisfy

$$u_\ell(s_1, \mathcal{X}) \approx u^*_\ell(s_1, \mathcal{X}) + \xi^* \cdot \mathbf{A}_{2\ell-1} \cdot \mathbf{y}$$
$$\approx u^*_\ell(s_1, \mathcal{X}) + \xi^* \sum_{j \neq \ell} N^j \big(A_{2\ell-1,2j-1} \cdot x_{j,1} + A_{2\ell-1,2j} \cdot x_{j,2}\big) \tag{3.9}$$

$$u_\ell(s_2, \mathcal{X}) \approx u^*_\ell(s_2, \mathcal{X}) + \xi^* \cdot \mathbf{A}_{2\ell} \cdot \mathbf{y}$$
$$\approx u^*_\ell(s_2, \mathcal{X}) + \xi^* \sum_{j \neq \ell} N^j \big(A_{2\ell,2j-1} \cdot x_{j,1} + A_{2\ell,2j} \cdot x_{j,2}\big) \tag{3.10}$$

where ξ^* is a parameter small enough to make sure that the resulting game is ξ-close to $\mathcal{G}^*_{n,N}$.

If one can perturb the payoff functions of players P_ℓ in $\mathcal{G}^*_{n,N}$ so that (3.9) and (3.10) hold for every ϵ-well-supported Nash equilibrium $\mathcal{X}$ of $\mathcal{G}_{\mathbf{A}}$, then the vector $\mathbf{y}$ obtained from $\mathcal{X}$ using (3.8) must be a $(1/n)$-well-supported equilibrium of $\mathbf{A}$. To see this, assume for contradiction that

$$\mathbf{A}_{2\ell-1} \cdot \mathbf{y} > \mathbf{A}_{2\ell} \cdot \mathbf{y} + 1/n \tag{3.11}$$

but $y_{2\ell} > 0$. Using (3.11), (3.9), and (3.10), we have $u_\ell(s_1, \mathcal{X})$ is bigger than $u_\ell(s_2, \mathcal{X})$ by ξ^*/n (assuming that errors hidden in both (3.9) and (3.10) are negligible). As long as our choice of ξ^* satisfies $\xi^*/n > \epsilon$ we must have $x_{\ell,2} = 0$ and thus, $y_{2\ell} = 0$ from (3.8).

However, perturbing the generalized radix game so that (3.9) and (3.10) hold is challenging. While

$$\sum_{j \neq \ell} N^j \big(A_{2\ell-1,2j-1} \cdot x_{j,1} + A_{2\ell-1,2j} \cdot x_{j,2}\big) \quad \text{and} \quad \sum_{j \neq \ell} N^j \big(A_{2\ell,2j-1} \cdot x_{j,1} + A_{2\ell,2j} \cdot x_{j,2}\big) \tag{3.12}$$

are merely two linear forms of $(x_{j,1}, x_{j,2} : j \neq \ell)$ from $\mathcal{X}$, they are extremely difficult to obtain due to the nature of anonymous games: the expected payoff of player P_ℓ is

$$u_\ell(\sigma, \mathcal{X}) = \sum_{\mathbf{k} \in K} \mathsf{payoff}_\ell(\sigma, \mathbf{k}) \cdot \boldsymbol{Pr}_{\mathcal{X}}[P_\ell, \mathbf{k}], \tag{3.13}$$

a linear form of $\boldsymbol{Pr}_{\mathcal{X}}[P_\ell, \mathbf{k}]$, the probability of P_ℓ seeing histogram $\mathbf{k}$ given $\mathcal{X}$. As each

$\boldsymbol{Pr}_{\mathcal{X}}[P_\ell, \mathbf{k}]$ is a highly complex and symmetric expression of variables in $\mathcal{X}$, it is not clear how one can extract from (3.13) the desired linear forms of (3.12).

This is where the fact that $x_{i,1} + x_{i,2} \approx \delta^i$ helps us tremendously. (Recall that this holds as long as the generalized radix game $\mathcal{G}^*_{n,N}$ and $\mathcal{G}_{\mathbf{A}}$ are ξ-close.) The core of the construction of $\mathcal{G}_{\mathbf{A}}$ uses the following key technical lemma which we refer to as the *estimation lemma*. It shows that under any mixed strategy profile $\mathcal{X} = (x_{i,1}, x_{i,2}, y, z : i \in [n])$ such that $x_{i,1} + x_{i,2} \approx \delta^i$, there is indeed a linear form of $\boldsymbol{Pr}_{\mathcal{X}}[P_\ell, \mathbf{k}]$ that gives us a close approximation of $x_{j,1}$ (or $x_{j,2}$), $j \neq \ell$, and its coefficients can be computed in polynomial time in n. We delay its proof to Section 3.6.

Lemma 3.5.1 (Estimation Lemma)**.** *Let $N = 2^n$ and $\lambda = 2^{-n^3}$. Given $\ell \in [n]$ and $j \neq \ell \in [n]$ one can compute in polynomial time in n vectors $\mathbf{B}^{[\ell,j]}, \mathbf{C}^{[\ell,j]}$ of length $|K|$ (indexed by $\mathbf{k} \in K$) such that every mixed strategy profile $\mathcal{X} = (x_{i,1}, x_{i,2}, y, z : i \in [n])$ with $x_{i,1} + x_{i,2} = \delta^i \pm \lambda$ for all i satisfies*

$$\sum_{\mathbf{k}\in K} B^{[\ell,j]}_{\mathbf{k}} \cdot \boldsymbol{Pr}\mathcal{X}P_\ell, \mathbf{k} = x_{j,1} \pm O\left(j^2\delta^{j+1}\right) \quad \textit{and} \quad \sum_{\mathbf{k}\in K} C^{[\ell,j]}_{\mathbf{k}} \cdot \boldsymbol{Pr}\mathcal{X}P_\ell, \mathbf{k} = x_{j,2} \pm O\left(j^2\delta^{j+1}\right).$$

Moreover, the absolute value of each entry of $\mathbf{B}^{[\ell,j]}$ and $\mathbf{C}^{[\ell,j]}$ is at most N^{n^2}.

With the estimation lemma in hand we can derive linear forms of $\boldsymbol{Pr}_{\mathcal{X}}[P_\ell, \mathbf{k}]$ that are close approximations of the two linear forms of $(x_{j,1}, x_{j,2} : j \neq \ell)$ in (3.12). We then use the coefficients of these linear forms of $\boldsymbol{Pr}_{\mathcal{X}}[P_\ell, \mathbf{k}]$ to perturb $\mathcal{G}^*_{n,N}$ and wrap up the construction of $\mathcal{G}_{\mathbf{A}}$.

3.5.2 Construction of Anonymous Game $\mathcal{G}_{\mathbf{A}}$

Let $\mathbf{A} \in [0,1]^{2n\times 2n}$ denote the input polymatrix game. We need the following parameters:

$$N = 2^n,\ \ \delta = 1/N = 2^{-n},\ \ \lambda = 2^{-n^3},\ \ \xi = 2^{-n^4},\ \ \xi^* = 2^{-n^5} \ \text{ and } \ \epsilon = 2^{-n^6}.$$

We remark that we do not attempt to optimize the parameters here but rather set them in different scales to facilitate the analysis later.

Game 3 (Construction of $\mathcal{G}_{\mathbf{A}}$)**.** We use the polynomial-time algorithm promised in the Estimation Lemma to compute $\mathbf{B}^{[\ell,j]}$ and $\mathbf{C}^{[\ell,j]}$, for all $\ell \in [n]$ and $j \neq \ell \in [n]$.

Starting with the generalized radix game $\mathcal{G}^*_{n,N}$, we modify payoff functions of players $P_1, \ldots, P_n$ as follows *(*payoff functions of Q and R remain unchanged*)*. Let $\textsf{payoff}^*_\ell$ denote the payoff function of P_ℓ in $\mathcal{G}^*_{n,N}$. Then for each player P_ℓ and each histogram $\mathbf{k} \in K$, we set

$$\textsf{payoff}_\ell(s_1, \mathbf{k}) = \textsf{payoff}^*_\ell(s_1, \mathbf{k}) + \xi^* \sum_{j \neq \ell} N^j \left(A_{2\ell-1,2j-1} \cdot B^{[\ell,j]}_{\mathbf{k}} + A_{2\ell-1,2j} \cdot C^{[\ell,j]}_{\mathbf{k}} \right)$$

$$\textsf{payoff}_\ell(s_2, \mathbf{k}) = \textsf{payoff}^*_\ell(s_2, \mathbf{k}) + \xi^* \sum_{j \neq \ell} N^j \left(A_{2\ell,2j-1} \cdot B^{[\ell,j]}_{\mathbf{k}} + A_{2\ell,2j} \cdot C^{[\ell,j]}_{\mathbf{k}} \right),$$

and keep all other payoffs of P_ℓ the same *(*i.e., $\textsf{payoff}_\ell(\sigma, \mathbf{k}) = \textsf{payoff}^*_\ell(\sigma, \mathbf{k})$ for all $\sigma \notin \{s_1, s_2\}$*)*.

A few properties of $\mathcal{G}_{\mathbf{A}}$ then follow directly from its construction. First, observe that entries of $\mathbf{A}$ lie in $[0, 1]$ and entries of $\mathbf{B}^{[\ell,j]}$ and $\mathbf{C}^{[\ell,j]}$ have absolute values at most $N^{n^2} = 2^{n^3}$. We have

Property 3.5.2. Given $\mathbf{A} \in [0, 1]^{2n \times 2n}$, $\mathcal{G}_{\mathbf{A}}$ is an anonymous game ξ-close to $\mathcal{G}^*_{n,N}$ where $\xi = 2^{-n^4}$.

By Lemma 3.4.2, an ϵ-well-supported Nash equilibrium of $\mathcal{G}_{\mathbf{A}}$ is fully described by a $(2n + 2)$-tuple $\mathcal{X} = (x_{i,1}, x_{i,2}, y, z : i \in [n])$, where P_i plays strategies s_1, s_2 and t with probabilities $x_{i,1}, x_{i,2}$ and $1 - x_{i,1} - x_{i,2}$, respectively. We also get the following corollary from Lemma 3.4.3.

Corollary 3.5.3. *Every ϵ-well-supported equilibrium $\mathcal{X} = (x_{i,1}, x_{i,2}, y, z : i \in [n])$ of $\mathcal{G}_{\mathbf{A}}$ satisfies*

$$x_{i,1} + x_{i,2} = \delta^i \pm \lambda, \quad \textit{for all } i \in [n].$$

Therefore, the conditions of the estimation lemma are met. It follows that

Property 3.5.4. Given an ϵ-well-supported equilibrium $\mathcal{X}$ of $\mathcal{G}_{\mathbf{A}}$, the expected payoffs of P_ℓ satisfy

$$u_\ell(s_1, \mathcal{X}) = u^*_\ell(s_1, \mathcal{X}) + \xi^* \sum_{j \neq \ell} N^j \big(A_{2\ell-1,2j-1} \cdot x_{j,1} + A_{2\ell-1,2j} \cdot x_{j,2}\big) \pm O(n^3 \xi^* \delta) \quad \text{and}$$

$$u_\ell(s_2, \mathcal{X}) = u^*_\ell(s_2, \mathcal{X}) + \xi^* \sum_{j \neq \ell} N^j \big(A_{2\ell,2j-1} \cdot x_{j,1} + A_{2\ell,2j} \cdot x_{j,2}\big) \pm O(n^3 \xi^* \delta).$$

3.5.3 Correctness of the Reduction

We are now ready to show that, given an ϵ-well-supported Nash equilibrium $\mathcal{X}$ of $\mathcal{G}_{\mathbf{A}}$, the vector $\mathbf{y}$ derived from $\mathcal{X}$ using (3.8) is a $(1/n)$-well-supported Nash equilibrium of the polymatrix game $\mathbf{A}$.

Lemma 3.5.5. *Let $\mathcal{X} = (x_{i,1}, x_{i,2}, y, z : i \in [n])$ be an ϵ-well supported Nash equilibrium of $\mathcal{G}_{\mathbf{A}}$. Then the vector $\mathbf{y} \in [0,1]^{2n}$ derived from $\mathcal{X}$ using* (3.8) *is a $(1/n)$-well-supported Nash equilibrium of $\mathbf{A}$.*

Proof. Firstly, note that $x_{i,1} + x_{i,2} > 0$ so $\mathbf{y}$ is well defined and satisfies $y_{2i-1} + y_{2i} = 1$ for all i.

Assume towards a contradiction that $\mathbf{y}$ derived from $\mathcal{X}$ using (3.8) is not a $(1/n)$-well-supported Nash equilibrium of $\mathbf{A}$, i.e., there is a player $\ell \in [n]$ such that, without loss of generality,

$$\mathbf{A}_{2\ell-1} \cdot \mathbf{y} > \mathbf{A}_{2\ell} \cdot \mathbf{y} + 1/n \tag{3.14}$$

but $y_{2\ell} > 0$, which in turn implies that $x_{\ell,2} > 0$.

Since $x_{j,1} + x_{j,2} = \delta^j \pm \lambda$, we have

$$y_{2j-1} = \frac{x_{j,1}}{x_{j,1} + x_{j,2}} = N^j x_{j,1} \pm O(N^{2j}\lambda) = N^j x_{j,1} \pm O\left(N^{2n}\lambda\right).$$

Similarly we also have $y_{2j} = N^j x_{j,2} \pm O(N^{2n}\lambda)$. Combining these with Property 3.5.4, we have

$$u_\ell(s_1, \mathcal{X}) = u^*_\ell(s_1, \mathcal{X}) + \xi^* \cdot \mathbf{A}_{2\ell-1} \cdot \mathbf{y} \pm \left(O(n^3\xi^*\delta) + O(n\xi^* N^{2n}\lambda)\right) \quad \text{and}$$
$$u_\ell(s_2, \mathcal{X}) = u^*_\ell(s_2, \mathcal{X}) + \xi^* \cdot \mathbf{A}_{2\ell} \cdot \mathbf{y} \pm \left(O(n^3\xi^*\delta) + O(n\xi^* N^{2n}\lambda)\right). \tag{3.15}$$

By our choices of parameters, $n\xi^* N^{2n}\lambda \ll n^3\xi^*\delta$ so the former can be absorbed into the latter.

Combining (3.14) and (3.15) (as well as the fact that $u^*_\ell(s_1, \mathcal{X}) = u^*_\ell(s_2, \mathcal{X})$ because the payoffs of s_1 and s_2 are exactly the same in the generalized radix game $\mathcal{G}^*_{n,N}$), we have

$$u_\ell(s_1, \mathcal{X}) - u_\ell(s_2, \mathcal{X}) \geq \xi^* \left(\mathbf{A}_{2\ell-1} \cdot \mathbf{y} - \mathbf{A}_{2\ell} \cdot \mathbf{y}\right) - O\left(n^3\xi^*\delta\right) \geq \xi^*/n - O\left(n^3\xi^*\delta\right) > \epsilon,$$

for sufficiently large n, by our choices of parameters δ, ξ^* and ϵ. It then follows that $x_{\ell,2} = 0$, since $\mathcal{X}$ is assumed to be an ϵ-well-supported Nash equilibrium of $\mathcal{G}_{\mathbf{A}}$, contradicting with $y_{2\ell} > 0$. This finishes the proof. □

3.5.4 Proof of the Hardness Part of Theorem 3.2.1

From our definitions of $\mathcal{G}^*_{n,N}$ and $\mathcal{G}_{\mathbf{A}}$, it is clear that all payoffs of $\mathcal{G}_{\mathbf{A}}$ are in $[-1, 3]$. Using standard arguments (invariance of Nash equilibria under shifting and scaling), we can easily see that given an anonymous game $\mathcal{G} = (n, \alpha, \{\mathsf{payoff}_p\})$ such that all payoffs are in the interval $[a, b]$, where $a, b \in \mathbb{R}$ and $a < b$, a mixed strategy profile $\mathcal{X}$ is an $(b-a)\epsilon$-well-supported equilibrium of $\mathcal{G}$ if and only if $\mathcal{X}$ is an ϵ-well-supported equilibrium of $\mathcal{G}' = (n, \alpha, \{\mathsf{payoff}'_p\})$, where

$$\mathsf{payoff}'_p(\sigma, \mathbf{k}) = \frac{\mathsf{payoff}_p(\sigma, \mathbf{k}) - a}{b - a}. \tag{3.16}$$

The new game $\mathcal{G}'$ now has all payoffs from in $[0, 1]$.

As a result, we can construct $\mathcal{G}'_{\mathbf{A}}$ from $\mathcal{G}_{\mathbf{A}}$ in polynomial time such that all payoffs of $\mathcal{G}'_{\mathbf{A}}$ lie in $[0, 1]$, and Lemma 3.5.5 holds for all $(\epsilon/4)$-well-supported Nash equilibria of $\mathcal{G}'_{\mathbf{A}}$. It follows that

Corollary 3.5.6. *Fix any $\alpha \geq 7$. The problem of finding a $2^{-(n^6+2)}$-well-supported Nash equilibrium of an anonymous game with α actions and $[0, 1]$ payoffs is PPAD-hard.*

This can be further strengthened using a standard padding argument.

Lemma 3.5.7. *Fix any $\alpha \in \mathbb{N}$ and $a > b > 0$. There is a polynomial-time reduction from the problem of finding a (2^{-n^a})-well-supported equilibrium to that of finding a (2^{-n^b})-well-supported equilibrium, in an anonymous game with α actions and $[0, 1]$ payoffs.*

Proof. For convenience, we will refer to the problem of finding a (2^{-n^a})-well-supported equilibrium as problem A and the other as problem B.

Let $\mathcal{G} = (n, \alpha, \{\mathsf{payoff}_p\})$ denote an input anonymous game of problem A. We define a new game $\mathsf{pad}\mathcal{G} = (n^t, \alpha, \{\mathsf{payoff}'_p\})$ as follows, where $t = a/b > 1$ and thus, $n^t > n$. To this end, define a map $\phi : \mathbb{Z}^\alpha \to \mathbb{Z}^\alpha$ such that $\phi(k_1, \ldots, k_\alpha) = (k_1 - (n^t - n), k_2, \ldots, k_\alpha)$. We then define payoff functions of players $\{1, \ldots, n^t\}$ in $\mathsf{pad}\mathcal{G}$ as follows:

- For each $i > n$, the payoff function of player i is given by

$$\mathsf{payoff}'_i(\sigma, \mathbf{k}) = \begin{cases} 1 & \text{if } \sigma = 1 \\ 0 & \text{otherwise} \end{cases}$$

  So player i always plays strategy 1 in any ϵ-well-supported equilibrium with $\epsilon < 1$.

- The payoff of each player $i \in [n]$ is given by

$$\mathsf{payoff}'_i(\sigma, \mathbf{k}) = \begin{cases} \mathsf{payoff}_i(\sigma, \phi(\mathbf{k})) & \text{if } k_1 \geq n^t - n \\ 0 & \text{otherwise} \end{cases}$$

Note that in any ϵ-well-supported equilibrium with $\epsilon < 1$, the latter case never occurs.

By the definition of $\mathsf{pad}\mathcal{G}$, it is easy to show that $\mathcal{X}$ is an ϵ-well-supported equilibrium in $\mathsf{pad}\mathcal{G}$, for some $\epsilon < 1$, iff 1) each player $i > n$ plays strategy 1 with probability 1 and 2) the mixed strategy profile of the first n players in $\mathcal{X}$ is an ϵ-well-supported equilibrium of $\mathcal{G}$. As a result, a solution to $\mathsf{pad}\mathcal{G}$ as an input of problem B must be an ϵ-approximate equilibrium of $\mathcal{G}$ with $\epsilon = 2^{-(n^t)^b} = 2^{-n^a}$. As $\mathsf{pad}\mathcal{G}$ can be constructed from $\mathcal{G}$ in polynomial time, this finishes the proof of the lemma. □

Combining Corollary 3.5.6 and Lemma 3.5.7, we have

Corollary 3.5.8. *Fix any $\alpha \geq 7$ and $c > 0$. The problem of finding a (2^{-n^c})-well-supported Nash equilibrium in an anonymous game with α actions and $[0, 1]$ payoffs is PPAD-hard.*

To prove the hardness part of Theorem 3.2.1, we next give a polynomial-time algorithm to compute a well-supported equilibrium from an approximate equilibrium.

Lemma 3.5.9 (Approximate to Well-Supported Nash Equilibria)**.** *Let $\mathcal{G} = (n, \alpha, \{\mathsf{payoff}_p\})$ be an anonymous game with payoffs from $[0, 1]$. Given an $\epsilon^2/(16\alpha n)$-approximate Nash equilibrium $\mathcal{X}$ of $\mathcal{G}$, one can compute in polynomial time an ϵ-well-supported Nash equilibrium $\mathcal{Y}$ of $\mathcal{G}$.*

Proof. Let $\mathcal{X} = (\mathbf{x}_i : i \in [n])$ be an ϵ'-approximate Nash equilibrium of $\mathcal{G}$, with $\epsilon' = \epsilon^2/(16\alpha n)$. For each player $i \in [n]$, we have for any mixed strategy $\mathbf{x}'_i$,

$$u_i(\mathbf{x}'_i, \mathcal{X}_{-i}) \leq u_i(\mathcal{X}) + \epsilon', \tag{3.17}$$

where we let $u_i(\mathbf{x}'_i, \mathcal{X}_{-i})$ denote the expected payoff of player i when she plays $\mathbf{x}'_i$ and other players play $\mathcal{X}_{-i}$. Let σ_i be a strategy with the highest expected payoff for player i (with respect to $\mathcal{X}_{-i}$):

$$u_i(\sigma_i, \mathcal{X}) = \max_{k \in [\alpha]} u_i(k, \mathcal{X}),$$

and let $J_i = \{j : u_i(\sigma_i, \mathcal{X}) \geq u_i(j, \mathcal{X}) + \epsilon/2\}$. We then define a mixed strategy $\mathbf{y}_i$ for player i using $\mathbf{x}_i$, σ_i and J_i as follows: Set $y_{i,j} = 0$ for all $j \in J_i$, and set

$$y_{i,\sigma_i} = x_{i,\sigma_i} + \sum_{j \in J_i} x_{i,j}.$$

All other entries of $\mathbf{y}_i$ are the same as $\mathbf{x}_i$. As $\mathbf{y}_i$ increases the expected payoff of player i by at least

$$(\epsilon/2) \cdot \sum_{j \in J_i} x_{i,j},$$

we have from (3.17) that $\sum_{j \in J_i} x_{i,j} \leq 2\epsilon'/\epsilon$.

Repeating this for every player $i \in [n]$, we obtain a new mixed strategy profile $\mathcal{Y}$ (clearly $\mathcal{Y}$ can be computed in polynomial time given $\mathcal{X}$). We finish the proof of the lemma by showing that $\mathcal{Y}$ is indeed an ϵ-well-supported Nash equilibrium of $\mathcal{G}$. Below we write $\zeta = 2\epsilon'/\epsilon$.

First, by the definition of $\mathcal{Y}$, $|x_{i,j} - y_{i,j}| \leq \zeta$ for all i, j. Thus, for any pure strategy profile $\boldsymbol{s}_{-i}$,

$$\prod_{q \neq i} y_{q,s_q} \geq \prod_{q \neq i} \max\left\{0, x_{q,s_q} - \zeta\right\} \geq \prod_{q \neq i} x_{q,s_q} - \zeta \cdot \sum_{q \neq i} \prod_{p \notin \{i,q\}} x_{p,s_p} \quad \text{and}$$

$$\prod_{q \neq i} x_{q,s_q} \geq \prod_{q \neq i} \max\left\{0, y_{q,s_q} - \zeta\right\} \geq \prod_{q \neq i} y_{q,s_q} - \zeta \cdot \sum_{q \neq i} \prod_{p \notin \{i,q\}} y_{p,s_p}.$$

Since all payoffs are in $[0,1]$, we have for any player $i \in [n]$ and pure strategy $j \in [\alpha]$ that

$$\begin{aligned}
\left|u_i(j,\mathcal{Y}) - u_i(j,\mathcal{X})\right| &\leq \sum_{\boldsymbol{s}_{-i}\in[\alpha]^{n-1}} \left| \prod_{q\neq i} y_{q,s_q} - \prod_{q\neq i} x_{q,s_q} \right| \\
&\leq \zeta \sum_{\boldsymbol{s}_{-i}} \sum_{q\neq i} \prod_{p\notin\{i,q\}} x_{p,s_p} + \zeta \sum_{\boldsymbol{s}_{-i}} \sum_{q\neq i} \prod_{p\notin\{i,q\}} y_{p,s_p} \\
&= \zeta \sum_{q\neq i} \sum_{\boldsymbol{s}_{-i}} \prod_{p\notin\{i,q\}} x_{p,s_p} + \zeta \sum_{q\neq i} \sum_{\boldsymbol{s}_{-i}} \prod_{p\notin\{i,q\}} y_{p,s_p} \\
&\leq \alpha\zeta \sum_{q\neq i} \left(\sum_{s_r : r\notin\{i,q\}} \prod_{p\notin\{i,q\}} x_{p,s_p} \right) + \alpha\zeta \sum_{q\neq i} \left(\sum_{s_r : r\notin\{i,q\}} \prod_{p\notin\{i,q\}} y_{p,s_p} \right) \\
&= 2(n-1)\alpha\zeta.
\end{aligned}$$

This implies that for any pure strategies $j, k \in [\alpha]$ we have

$$\left|(u_i(j,\mathcal{X}) - u_i(k,\mathcal{X})) - (u_i(j,\mathcal{Y}) - u_i(k,\mathcal{Y}))\right| < \epsilon/2.$$

Therefore, the new mixed strategy profile $\mathcal{Y} = (\mathbf{y}_i : i \in [n])$ satisfies

$$u_i(j,\mathcal{Y}) < u_i(k,\mathcal{Y}) + \epsilon \;\Rightarrow\; u_i(j,\mathcal{X}) < u_i(k,\mathcal{X}) + \epsilon/2 \;\Rightarrow\; y_{i,j} = 0$$

for all i, j and k. This finishes the proof of the lemma. □

Fix any $\alpha \geq 7$ and $c > 0$. It then follows from Lemma 3.5.9 that the problem of finding a $(2^{-n^{c/2}})$ well-supported equilibrium in an anonymous game with α actions and $[0,1]$ payoffs is polynomial-time reducible to problem (α, c)-ANONYMOUS. As the former problem is PPAD-hard by Corollary 3.5.8, (α, c)-ANONYMOUS is PPAD-hard. The finishes the proof of the hardness part of Theorem 3.2.1.

3.6 Proof of the Estimation Lemma

We prove the estimation lemma (Lemma 3.5.1) in this section.

Recall that there are n main players $P_1, \ldots, P_n$, and they are only interested in three strategies $\{s_1, s_2, t\}$. For convenience we will refer to s_1 as strategy 1, s_2 as strategy 2, and t as strategy 3 in this section. Player P_i plays strategy $b \in [3]$ with probability $x_{i,b}$, and $\sum_b x_{i,b} = 1$. While $x_{i,b}$'s are unknown variables, by the assumption of the lemma we are guaranteed that

$$x_{i,1} + x_{i,2} = \delta^i \pm \lambda, \quad \text{where } \lambda = \delta^{n^2}. \tag{3.18}$$

Throughout this section we will *fix* two distinct integers $r, \ell \in [n]$, and the goal will be to derive an approximation of the unknown $x_{r,1}$ for P_ℓ using a linear form of the following probabilities:

$$\Big\{\boldsymbol{Pr}\big[k_1 = i, k_2 = j\big] : i, j \in [0:n{-}1]\Big\}, \text{ where } \boldsymbol{Pr}\big[k_1 = i, k_2 = j\big] = \sum_{\substack{\mathbf{k}\in K \\ k_1=i, k_2=j}} \boldsymbol{Pr}_{\mathcal{X}}[P_\ell, \mathbf{k}], \tag{3.19}$$

and k_b denotes the random variable that counts players playing $b \in [3]$ other than player P_ℓ herself. We will show that coefficients in the desired linear form can be computed in polynomial time in n.

First we would like to give the reader some intuition for the rest of the section, by showing how one can get a good estimate of $x_{1,1}$ and $x_{2,1}$, assuming $\ell > 2$. We believe this to be useful for more easily understanding the rest of the section, but the reader should feel free to skip it, if desired.

Remark 3.6.1 (Informal)**.** As $N = 2^n$ is large, we have $x_{i,3} \approx 1$ for each i. This gives

$$\boldsymbol{Pr} k_1 = 1, k_2 = 0 \approx x_{1,1} + x_{2,1} + \cdots + x_{n,1} = x_{1,1} \pm O(\delta^2)$$

as $x_{i,1} \leq \delta^i + \lambda$. Similarly, $\boldsymbol{Pr} k_1 = 2, k_2 = 0 \approx x_{1,1}x_{2,1} \pm O(\delta^4)$. Using $x_{i,1} + x_{i,2} \approx \delta^i$,

we have

$$\boldsymbol{Pr}k_1 = k_2 = 1 \approx x_{1,1}(\delta^2 - x_{2,1}) + x_{2,1}(\delta - x_{1,1}) \pm O(\delta^4) = \delta^2 x_{1,1} + \delta x_{2,1} - 2x_{1,1}x_{2,1} \pm O(\delta^4).$$

Combining all three estimates, we have

$$N\Big(\boldsymbol{Pr}k_1 = k_2 = 1 + 2 \cdot \boldsymbol{Pr}k_1 = 2, k_2 = 0\Big) - \delta \cdot \boldsymbol{Pr}k_1 = 1, k_2 = 0 \approx x_{2,1} \pm O(\delta^3).$$

Since $x_{2,1} \leq \delta^2 + \lambda$, the linear form on the LHS gives us an additive approximation of $x_{2,1}$.

We need some notation in order to generalize and formalize this. Let $\mathcal{S} = [n] \setminus \{\ell\}$, the set of players observed by player P_ℓ. Let k_b, $b \in [3]$, denote the random variable that counts players from $\mathcal{S}$ that play strategy b. We write $\mathcal{L} = \{i \in \mathcal{S} : i \leq r\}$ and $m = |\mathcal{L}|$, i.e., $\mathcal{L} = [r]$ and $m = r$ if $\ell > r$, and $\mathcal{L} = [r] \setminus \{\ell\}$ and $m = r - 1$ if $\ell < r$. We start by understanding the following probabilities

$$\Big\{\boldsymbol{Pr}\big[k_1 = m - j, k_2 = j\big] : j \in [0 : m]\Big\}.$$

It will become clear that players from $\mathcal{S} \setminus \mathcal{L}$ have probabilities too small to significantly affect these probabilities (so their contribution will just be absorbed into the error term).

For $j \in [0 : m]$, let Δ_j denote the set of partitions of $\mathcal{S}$ into sets of size $m - j, j$ and $n - 1 - m$:

$$\Delta_j = \Big\{(\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3) : \mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3 \text{ are pairwise disjoint, } \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3 = \mathcal{S},\ |\mathcal{S}_1| = m - j,\ |\mathcal{S}_2| = j\Big\}.$$

So, by definition, we have

$$\boldsymbol{Pr}\big[k_1 = m - j, k_2 = j\big] = \sum_{(\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3) \in \Delta_j} \left(\prod_{i \in \mathcal{S}_1} x_{i,1} \prod_{i \in \mathcal{S}_2} x_{i,2} \prod_{i \in \mathcal{S}_3} x_{i,3}\right).$$

By (3.18) we can write $x_{i,1} + x_{i,2} = \delta^i + \lambda_i$ for some λ_i with $|\lambda_i| \le \lambda$. We can substitute to get

$$\boldsymbol{Pr}\big[k_1 = m - j, k_2 = j\big] = \sum_{(\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3)\in\Delta_j} \left(\prod_{i\in\mathcal{S}_1} x_{i,1} \prod_{i\in\mathcal{S}_2} (\delta^i + \lambda_i - x_{i,1}) \prod_{i\in\mathcal{S}_3} (1 - \delta^i - \lambda_i) \right). \tag{3.20}$$

Next, we split Δ_j into two sets Δ_j^* and Δ_j': $(\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3) \in \Delta_j$ is in Δ_j^* if $\mathcal{S}_1 \cup \mathcal{S}_2 = \mathcal{L}$; otherwise, it is in Δ_j'. This splits the sum in (3.20) into two sums accordingly, one over Δ_j^* and one over Δ_j'. We show in the following lemma that the contribution from the second sum is negligible.

Lemma 3.6.2. *Given the parameters in* (3.18)*, we have*

$$\sum_{(\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3)\in\Delta_j'} \left(\prod_{i\in\mathcal{S}_1} x_{i,1} \prod_{i\in\mathcal{S}_2} (\delta^i + \lambda_i - x_{i,1}) \prod_{i\in\mathcal{S}_3} (1 - \delta^i - \lambda_i) \right) = O\Big(\delta \prod_{i\in\mathcal{L}} \delta^i \Big).$$

Proof. Since all terms in the sum are nonnegative, it suffices to show that

$$\sum_{(\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3)\in\Delta_j'} \left(\prod_{i\in\mathcal{S}_1} x_{i,1} \prod_{i\in\mathcal{S}_2} (\delta^i + \lambda_i - x_{i,1}) \right) = O\Big(\delta \prod_{i\in\mathcal{L}} \delta^i \Big). \tag{3.21}$$

Fix a set $\mathcal{T} \subseteq \mathcal{S}$ such that $|\mathcal{T}| = m$ but $\mathcal{T} \neq \mathcal{L}$. We have

$$\prod_{i\in\mathcal{T}} (\delta^i + \lambda_i) = \prod_{i\in\mathcal{T}} \big(x_{i,1} + (\delta^i + \lambda_i - x_{i,1}) \big) = \sum_{\mathcal{S}_1\subseteq\mathcal{T}} \left(\prod_{i\in\mathcal{S}_1} x_{i,1} \prod_{i\in\mathcal{T}\setminus\mathcal{S}_1} (\delta^i + \lambda_i - x_{i,1}) \right).$$

Since every term on the RHS is nonnegative, we have

$$\sum_{\substack{\mathcal{S}_1\subseteq\mathcal{T} \\ |\mathcal{S}_1|=m-j}} \left(\prod_{i\in\mathcal{S}_1} x_{i,1} \prod_{i\in\mathcal{T}\setminus\mathcal{S}_1} (\delta^i + \lambda_i - x_{i,1}) \right) \le \prod_{i\in\mathcal{T}} (\delta^i + \lambda_i) = (1 + o(1)) \cdot \prod_{i\in\mathcal{T}} \delta^i,$$

given that $\lambda_i = \delta^{n^2}$ in (3.18). Let $h(\mathcal{T}) = \prod_{i\in\mathcal{T}} \delta^i$. To prove (3.21), it now suffices to show

that

$$\sum_{\substack{\mathcal{T}\subseteq\mathcal{S}\\ |\mathcal{T}|=m,\mathcal{T}\neq\mathcal{L}}} h(\mathcal{T}) = O\big(\delta\cdot h(\mathcal{L})\big) = O\bigg(\delta\prod_{i\in\mathcal{L}}\delta^i\bigg).$$

For this purpose, notice that $h(\mathcal{T}) \leq \delta\cdot h(\mathcal{L})$ for any $\mathcal{T}$ such that $\mathcal{T}\subseteq\mathcal{S}$, $|\mathcal{T}| = m$, but $\mathcal{T}\neq\mathcal{L}$. It is also easy to see that there is at most one $\mathcal{T}$ such that $h(\mathcal{T}) = \delta\cdot h(\mathcal{L})$. Because every other $\mathcal{T}$ has $h(\mathcal{T}) \leq \delta^2\cdot h(\mathcal{L})$ and the total number of $\mathcal{T}$'s is at most $2^{n-1} = N/2$, we have

$$\sum_{\mathcal{T}} h(\mathcal{T}) \leq \delta\cdot h(\mathcal{L}) + (N/2)\cdot\delta^2\cdot h(\mathcal{L}) = O(\delta\cdot h(\mathcal{L})),$$

as $\delta = 1/N$. This finishes the proof of the lemma. □

Combining (3.20) and Lemma 3.6.2, we have

$$\boldsymbol{Pr}\big[k_1 = m-j, k_2 = j\big] = \sum_{\substack{\mathcal{S}_1\subseteq\mathcal{L}\\ |\mathcal{S}_1|=m-j}} \left(\prod_{i\in\mathcal{S}_1} x_{i,1}\prod_{i\in\mathcal{L}\setminus\mathcal{S}_1}(\delta^i+\lambda_i-x_{i,1})\prod_{i\notin\mathcal{L}}(1-\delta^i-\lambda_i)\right) \pm O(\delta\cdot h(\mathcal{L})).$$

The next lemma further simplifies this estimate by absorbing all the λ_i's into the error term.

Lemma 3.6.3. *Given the parameters in* (3.18)*, we have*

$$\boldsymbol{Pr}k_1 = m-j, k_2 = j = \sum_{\substack{\mathcal{S}_1\in\mathcal{L}\\ |\mathcal{S}_1|=m-j}} \left(\prod_{i\in\mathcal{S}_1} x_{i,1}\prod_{i\in\mathcal{L}\setminus\mathcal{S}_1}(\delta^i-x_{i,1})\prod_{i\notin\mathcal{L}}(1-\delta^i)\right) \pm O(\delta\cdot h(\mathcal{L})).$$

Proof. First the number of $\mathcal{S}_1$'s is at most $2^{n-1} < N$. Further, fixing an $\mathcal{S}_1$ and multiplying out

$$\prod_{i\in\mathcal{S}_1} x_{i,1}\prod_{i\in\mathcal{L}\setminus\mathcal{S}_1}(\delta^i+\lambda_i-x_{i,1})\prod_{i\notin\mathcal{L}}(1-\delta^i-\lambda_i)$$

will yield $3^j\cdot 3^{n-1-m} \leq 3^{n-1} < N^2$ many terms. The absolute value of each term with at least one λ_i must be less than or equal to λ because all factors are less than or equal to 1. There are at most N^2 many such terms, for each $\mathcal{S}_1$, and there are at most N different $\mathcal{S}_1$'s.

Using $N^3\lambda \ll \delta h(\mathcal{L})$ by (3.18), we can absorb all terms with at least one λ_i into the error term $O(\delta \cdot h(\mathcal{L}))$. $\square$

Using Lemma 3.6.3 and the fact that $\prod_{i\notin\mathcal{L}}(1-\delta^i) > 1/2$ as $\delta = 1/2^n$, we have

$$\left(\prod_{i\notin\mathcal{L}}(1-\delta^i)\right)^{-1} \boldsymbol{Pr}\big[k_1 = m-j, k_2 = j\big] = \sum_{\substack{\mathcal{S}_1\subseteq\mathcal{L}\\ |\mathcal{S}_1|=m-j}} \left(\prod_{i\in\mathcal{S}_1} x_{i,1} \prod_{i\in\mathcal{L}\setminus\mathcal{S}_1} (\delta^i - x_{i,1})\right) \pm O(\delta\cdot h(\mathcal{L})).$$

To understand the RHS better, we define a polynomial P_d^m for each $d \in [0:m]$ to be

$$P_d^m = \sum_{\mathcal{T}\subseteq\mathcal{L}, |\mathcal{T}|=d} \left(\prod_{i\in\mathcal{T}} x_{i,1} \prod_{i\in\mathcal{L}\setminus\mathcal{T}} \delta^i\right),$$

and prove the following lemma that establishes a connection between them.

Lemma 3.6.4. *Given P_d^m defined above, we have*

$$\sum_{\substack{\mathcal{S}_1\subseteq\mathcal{L}\\ |\mathcal{S}_1|=m-j}} \left(\prod_{i\in\mathcal{S}_1} x_{i,1} \prod_{i\in\mathcal{L}\setminus\mathcal{S}_1} (\delta^i - x_{i,1})\right) = \sum_{i=0}^{j} (-1)^i \cdot \binom{m-j+i}{m-j} \cdot P_{m-j+i}^m \tag{3.22}$$

Proof. Note that every monomial that appears on the two sides of (3.22) has the form $\prod_{i\in\mathcal{T}} x_{i,1}$ for some $\mathcal{T} \subseteq \mathcal{L}$ with $|\mathcal{T}| = d \geq m-j$. Fix such a $\mathcal{T}$. The coefficient of $\prod_{i\in\mathcal{T}} x_{i,1}$ on RHS of (3.22) is

$$(-1)^{d-m+j} \cdot \binom{d}{m-j} \cdot \prod_{i\in\mathcal{L}\setminus\mathcal{T}} \delta^i.$$

On the other hand, for an $\mathcal{S}_1 \subseteq \mathcal{L}$ with $|\mathcal{S}_1| = m-j$, we have

$$\prod_{i\in\mathcal{S}_1} x_{i,1} \prod_{i\in\mathcal{L}\setminus\mathcal{S}_1} (\delta^i - x_{i,1}) = \sum_{\mathcal{S}'\subseteq\mathcal{L}\setminus\mathcal{S}_1} \left(\prod_{i\in\mathcal{S}_1} x_{i,1} \prod_{i\in\mathcal{S}'} (-x_{i,1}) \prod_{i\in\mathcal{L}\setminus\{\mathcal{S}_1\cup\mathcal{S}'\}} \delta^i\right).$$

Hence, $\prod_{i\in\mathcal{T}} x_{i,1}$ occurs exactly once in this sum if and only if $\mathcal{S}_1 \subseteq \mathcal{T}$, and will take the form

$$\prod_{i\in\mathcal{S}_1} x_{i,1} \prod_{i\in\mathcal{T}\setminus\mathcal{S}_1} (-x_{i,1}) \prod_{i\in\mathcal{L}\setminus\mathcal{T}} \delta^i = (-1)^{d-m+j} \prod_{i\in\mathcal{T}} x_{i,1} \prod_{i\in\mathcal{L}\setminus\mathcal{T}} \delta^i.$$

Further, there are $\binom{d}{m-j}$ many $\mathcal{S}_1$ such that $\mathcal{S}_1 \subseteq \mathcal{T}$ and $|\mathcal{S}_1| = m - j$. The lemma is proven. □

Combining Lemma 3.6.3 and 3.6.4, we immediately get the following corollary:

Corollary 3.6.5. *For any $j \in [0 : m]$, we have*

$$\left(\prod_{i\notin\mathcal{L}}(1-\delta^i)\right)^{-1} \boldsymbol{Pr}k_1 = m-j, k_2 = j = \sum_{i=0}^{j} (-1)^i \cdot \binom{m-j+i}{m-j} \cdot P^m_{m-j+i} \pm O\big(\delta \cdot h(\mathcal{L})\big).$$

Taking a step back, we have derived a set of linear equations that hold with high precision over $\boldsymbol{Pr}[k_1 = m, k_2 = 0], \ldots, \boldsymbol{Pr}[k_1 = 0, k_2 = m]$ and $P^m_m, \ldots, P^m_0$. This then allows us to attain a close approximation for P^m_1, using a linear form of the m probabilities. Note that

$$P^m_1 = \sum_{i\in\mathcal{L}} x_{i,1} \prod_{j\in\mathcal{L}\setminus\{i\}} \delta^j = h(\mathcal{L}) \cdot \sum_{i\in\mathcal{L}} N^i \cdot x_{i,1} \tag{3.23}$$

is a linear form of the $x_{i,1}$'s, $i \in \mathcal{L}$, including $x_{r,1}$ (recall that r is the largest integer in $\mathcal{L}$). So from here, it will be straightforward to get an approximation of $x_{r,1}$.

The next lemma gives us a linear form to approximate P^m_1.

Lemma 3.6.6. *The m probabilities and P^m_1 satisfy*

$$\left(\prod_{i\notin\mathcal{L}}(1-\delta^i)\right)^{-1} \sum_{j=1}^{m} j \cdot \boldsymbol{Pr}k_1 = j, k_2 = m-j = P^m_1 \pm O\big(m^2\delta \cdot h(\mathcal{L})\big).$$

Proof. By Corollary 3.6.5 (and replacing j init by $m - j$), we see that it suffices to show that

$$P^m_1 = \sum_{j=1}^{m} j \cdot \left(\sum_{i=0}^{m-j} (-1)^i \cdot \binom{j+i}{j} \cdot P^m_{j+i}\right). \tag{3.24}$$

Consider P_d^m for some $d \in [m]$. P_d^m appears in the jth term on the RHS of (3.24) if and only if $d \geq j$, and when this is the case, the coefficient of P_d^m is

$$j \cdot (-1)^{d-j} \cdot \binom{d}{j}.$$

So the RHS of (3.24) is

$$\sum_{d=1}^{m} P_d^m \cdot \left(\sum_{j=1}^{d} (-1)^{d-j} \cdot j \cdot \binom{d}{j} \right).$$

For $d = 1$, the coefficient of P_1^m is clearly 1. For $d > 1$, using $j\binom{d}{j} = d\binom{d-1}{j-1}$ we have

$$\sum_{j=1}^{d} (-1)^{d-j} \cdot j \cdot \binom{d}{j} = d \cdot \sum_{j=1}^{d} (-1)^{d-j} \cdot \binom{d-1}{j-1} = d \cdot \sum_{j=0}^{d-1} (-1)^{d-j-1} \binom{d-1}{j} = 0.$$

This finishes the proof of the lemma. □

Lemma 3.6.6 gives us a linear form to approximate P_1^m. Denote this linear form by Y_m. Then for the special case when $\mathcal{L} = \{r\}$ (so r is the only integer in $\mathcal{L}$), we are done since P_1^m is exactly $x_{r,1}$, and we have attained a linear form that approximates $x_{r,1}$ with error $O(m^2\delta \cdot h(\mathcal{L}))$.

Otherwise suppose $|\mathcal{L}| > 1$. We use r' to denote the largest integer in $\mathcal{L}$ other than r and write $\mathcal{L}' = \{i \in \mathcal{S} : i \leq r'\}$ ($|\mathcal{L}'| = m - 1$). Repeating the same line of proof so far over $\mathcal{L}'$ and $m - 1$, we obtain a linear form of $\boldsymbol{Pr}[k_1 = m - 1 - j, k_2 = j]$, $j \in [0 : m - 1]$, denoted by Y_{m-1}, to approximate

$$P_1^{m-1} = \sum_{i \in \mathcal{L}'} x_{i,1} \prod_{j \in \mathcal{L}' \setminus \{i\}} \delta^j = h(\mathcal{L}') \cdot \sum_{i \in \mathcal{L}'} N^i \cdot x_{i,1} \tag{3.25}$$

with error $O(m^2\delta \cdot h(\mathcal{L}'))$. By the definition of P_1^m and P_1^{m-1} in (3.23) and (3.25), we have

$$x_{r,1} = \delta^r \left(\frac{P_1^m}{h(\mathcal{L})} - \frac{P_1^{m-1}}{h(\mathcal{L}')} \right).$$

As a result, we have obtained a linear form

$$\delta^r \left(\frac{Y_m}{h(\mathcal{L})} - \frac{Y_{m-1}}{h(\mathcal{L}')} \right) = x_{r,1} \pm O(m^2 \delta^{r+1}) \tag{3.26}$$

over $\boldsymbol{Pr}[k_1 = m-j, k_2 = j]$, $j \in [0:m]$ and $\boldsymbol{Pr}[k_{,1} = m-1-j, k_2 = j]$, $j \in [0:m-1]$.

Finally, it follows easily from our derivation of Y_m and Y_{m-1} that coefficients of this linear form can be computed in polynomial time in n, and every coefficient has absolute value at most N^{m^2}.

3.7 Membership in PPAD

In this section we show that (α, c)-ANONYMOUS is in PPAD for any constants $\alpha \in \mathbb{N}$ and $c > 0$, i.e. the problem of finding an ϵ-approximate equilibrium in an anonymous game $\mathcal{G} = (n, \alpha, \{\mathsf{payoff}_p\})$ with payoffs from $[0,1]$ is in PPAD, where $\epsilon = 1/2^{n^c}$. Below we use $\text{SIZE}(\mathcal{G})$ to denote the input size of an anonymous game $\mathcal{G}$, i.e., length of the binary representation of $\mathcal{G}$. We write $\text{SIZE}(a)$ to denote the length of the binary representation of a rational number a, and let $\text{SIZE}(\boldsymbol{a}) = \sum_i \text{SIZE}(a_i)$ for a rational vector $\boldsymbol{a}$ (e.g., a rational mixed strategy profile).

Fix constants $\alpha \in \mathbb{N}$ and $c > 0$. We show the membership of (α, c)-ANONYMOUS by reducing it to a "weak-approximation" fixed point problem [122] (see [122] for the difference between *weak* and *strong* approximations). Given $\mathcal{G} = (n, \alpha, \{\mathsf{payoff}_p\})$, we define a map $F : \Delta \to \Delta$ (this is the map commonly used to prove the existence of Nash equilibria, e.g., see [115]), where

$$\Delta = \left\{ (\mathbf{x}_i : i \in [n]) : \mathbf{x}_i \in \mathbb{R}_+^\alpha \text{ is a mixed strategy of player } i \in [n] \right\}$$

is the set of all mixed strategy profiles. For each $i \in [n]$ and $j \in [\alpha]$, the (i,j)th component

of F

$$F_{i,j}(\mathcal{X}) = \frac{x_{i,j} + \max\left(0, u_i(j, \mathcal{X}) - u_i(\mathcal{X})\right)}{1 + \sum_{k \in [\alpha]} \max\left(0, u_i(k, \mathcal{X}) - u_i(\mathcal{X})\right)}, \tag{3.27}$$

where $\mathcal{X} = (\mathbf{x}_i : i \in [n]) \in \Delta$ and $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,\alpha})$ for each $i \in [n]$.

Observe that F is continuous and maps Δ to itself. We also have

Property 3.7.1. The map F defined above is *polynomial-time computable*: Given a rational $\mathcal{X} \in \Delta$, $F(\mathcal{X})$ is rational and can be computed in polynomial time in $\text{SIZE}(\mathcal{G})$ and $\text{SIZE}(\mathcal{X})$.

Proof. This follows from the fact that there is a polynomial-time dynamic programming algorithm (see [149]) that computes $u_i(j, \mathcal{X})$, given $\mathcal{G}$ and $\mathcal{X}$. □

We say $\mathcal{X} \in \Delta$ is an *ϵ-approximate fixed point* of F if $\|F(\mathcal{X}) - \mathcal{X}\|_\infty \leq \epsilon$. We prove Lemma 3.7.2 in Section 3.7.1, showing that approximate fixed points of F are approximate Nash equilibria of $\mathcal{G}$.

Lemma 3.7.2. *Given $\mathcal{X} \in \Delta$ and $0 \leq \epsilon \leq 1$, if $\|F(\mathcal{X}) - \mathcal{X}\|_\infty \leq \epsilon$, then we have $u_i(j, \mathcal{X}) \leq u_i(\mathcal{X}) + \epsilon'$ for all players $i \in [n]$ and pure strategies $j \in [\alpha]$, where $\epsilon' = \alpha^2 \epsilon^{1/3}$.*

So to find an ϵ-approximate Nash equilibrium $\mathcal{X}$ of $\mathcal{G}$, it suffices to find an (ϵ^3/α^6)-approximate fixed point of F. Moreover, we show in Section 3.7.2 that F is *polynomially Lipschitz continuous*:

Lemma 3.7.3. *For all $\mathcal{X}, \mathcal{Y} \in \Delta$, we have*

$$\|F(\mathcal{X}) - F(\mathcal{Y})\|_\infty \leq 10 n \alpha^{n+2} \cdot \|\mathcal{X} - \mathcal{Y}\|_\infty.$$

Combining Property 3.7.1 and Lemma 3.7.3, it follows from Proposition 2.2 (Part 2) of [122] that given $\mathcal{G}$ and ϵ (in binary), the problem of finding an ϵ-approximate fixed point $\mathcal{X}$ of F is in PPAD. The PPAD membership of (α, c)-ANONYMOUS then follows from Lemma 3.7.2.

3.7.1 Proof of Lemma 3.7.2

For convenience, we write $\max_{i,k}(\mathcal{X}) = \max\left(0, u_i(k, \mathcal{X}) - u_i(\mathcal{X})\right)$ for $i \in [n]$ and $k \in [\alpha]$.

In the pursuit of a contradiction, assume that there exist a player $i \in [n]$ and an action $\ell \in [\alpha]$ such that $u_i(\ell, \mathcal{X}) > u_i(\mathcal{X}) + \epsilon'$. This, along with the fact that $\max_{i,k}(\mathcal{X}) \in [0, 1]$, implies that,

$$\epsilon' < \textstyle\sum_{k \in [\alpha]} \max_{i,k}(\mathcal{X}) \leq \alpha - 1. \tag{3.28}$$

We will show that cases $x_{i,\ell} \leq \alpha\epsilon^{1/3}$ and $x_{i,\ell} > \alpha\epsilon^{1/3}$ both result in the existence of a strategy $j \in [\alpha]$ such that $|F_{i,j}(\mathcal{X}) - x_{i,j}| > \epsilon$, contradicting our initial assumption.

Case 1: $x_{i,\ell} \leq \alpha\epsilon^{1/3}$. Apply (3.27), (3.28) and $\epsilon' = \alpha^2\epsilon^{1/3}$ to get

$$F_{i,\ell}(\mathcal{X}) > \frac{x_{i,\ell} + \epsilon'}{\alpha} \;\Rightarrow\; F_{i,\ell}(\mathcal{X}) - x_{i,\ell} > \frac{\epsilon' - (\alpha - 1)x_{i,\ell}}{\alpha} \geq \frac{\epsilon' - (\alpha - 1)\alpha\epsilon^{1/3}}{\alpha} = \epsilon^{1/3} \geq \epsilon.$$

Case 2: $x_{i,\ell} > \alpha\epsilon^{1/3}$. Let $J = \{j \in [\alpha] : u_i(j, \mathcal{X}) \leq u_i(\mathcal{X})\}$. We must have

$$\textstyle\sum_{j \in J} x_{i,j}\big(u_i(\mathcal{X}) - u_i(j, \mathcal{X})\big) \geq x_{i,\ell}\big(u_i(\ell, \mathcal{X}) - u_i(\mathcal{X})\big),$$

where $u_i(\mathcal{X}) - u_i(j, \mathcal{X}) \leq 1 - \epsilon'$, $u_i(\ell, \mathcal{X}) - u_i(\mathcal{X}) \geq \epsilon'$, and $x_{i,\ell} > \alpha\epsilon^{1/3}$. Therefore,

$$\textstyle\sum_{j \in J} x_{i,j} \geq \dfrac{\alpha\epsilon'\epsilon^{1/3}}{1 - \epsilon'},$$

which implies that there exists some strategy $j \in J$ such that $x_{i,j} \geq \epsilon'\epsilon^{1/3}/(1 - \epsilon')$.

Apply (3.27) and (3.28) to get $F_{i,j}(\mathcal{X}) < x_{i,j}/(1+\epsilon')$, which implies that

$$\left|F_{i,j}(\mathcal{X}) - x_{i,j}\right| > \frac{\epsilon' x_{i,j}}{1+\epsilon'} \geq \frac{(\epsilon')^2 \epsilon^{1/3}}{(1-\epsilon')(1+\epsilon')} \geq \alpha^4 \epsilon \geq \epsilon.$$

This finishes the proof of Lemma 3.7.2.

3.7.2 Proof of Lemma 3.7.3

As $\mathcal{X} - \mathcal{Y}$ is of length $n\alpha$, we have $\|\mathcal{X} - \mathcal{Y}\|_1 \leq n\alpha \cdot \|\mathcal{X} - \mathcal{Y}\|_\infty$. Thus, it suffices to show that

$$\|F(\mathcal{X}) - F(\mathcal{Y})\|_\infty \leq 16\alpha^{n+1}\|\mathcal{X} - \mathcal{Y}\|_1.$$

Fix $i \in [n]$ and $j \in [\alpha]$. We have

$$\left|F_{i,j}(\mathcal{X}) - F_{i,j}(\mathcal{Y})\right| = \left| \frac{x_{i,j} + \max_{i,j}(\mathcal{X})}{1 + \sum_{k\in[\alpha]} \max_{i,k}(\mathcal{X})} - \frac{y_{i,j} + \max_{i,j}(\mathcal{Y})}{1 + \sum_{k\in[\alpha]} \max_{i,k}(\mathcal{Y})} \right|.$$

Multiplying the terms in the RHS to get a common denominator, which is clearly ≥ 1, we get

$$\left|F_{i,j}(\mathcal{X}) - F_{i,j}(\mathcal{Y})\right| \leq \left|x_{i,j} - y_{i,j}\right| + \left| x_{i,j} \sum_{k\in[\alpha]} \max_{i,k}(\mathcal{Y}) - y_{i,j} \sum_{k\in[\alpha]} \max_{i,k}(\mathcal{X}) \right| \tag{3.29}$$

$$+ \left| \max_{i,j}(\mathcal{X}) - \max_{i,j}(\mathcal{Y}) \right| + \tag{3.30}$$

$$\left| \max_{i,j}(\mathcal{X}) \sum_{k\in[\alpha]} \max_{i,k}(\mathcal{Y}) - \max_{i,j}(\mathcal{Y}) \sum_{k\in[\alpha]} \max_{i,k}(\mathcal{X}) \right|.$$

To bound $|F_{i,j}(\mathcal{X}) - F_{i,j}(\mathcal{Y})|$, we shall use the following simple trick several times in the rest of the proof. If $a_1, a_2, b_1, b_2 \in [0,1]$, then we have

$$|a_1 a_2 - b_1 b_2| = |(a_1 - b_1)a_2 + b_1(a_2 - b_2)| \leq |a_1 - b_1| + |a_2 - b_2|,$$

which easily extends to

$$|a_1 \cdots a_n - b_1 \cdots b_n| \leq |a_1 - b_1| + \cdots + |a_n - b_n|,$$

when all the a_i's and b_i's are in $[0, 1]$.

Now we come back to (3.29). By the definition of $\max_{i,j}(\mathcal{X})$, we have

$$\begin{aligned}\Big|\max_{i,j}(\mathcal{X}) - \max_{i,j}(\mathcal{Y})\Big| &\leq \big|(u_i(j, \mathcal{X}) - u_i(\mathcal{X})) - (u_i(j, \mathcal{Y}) - u_i(\mathcal{Y}))\big| \\ &\leq \big|u_i(j, \mathcal{X}) - u_i(j, \mathcal{Y})\big| + \big|u_i(\mathcal{X}) - u_i(\mathcal{Y})\big|.\end{aligned}$$

As $\mathcal{X}, \mathcal{Y} \in \Delta$ we have $x_{i,j}, y_{i,j} \in [0, 1]$. Since all payoffs of $\mathcal{G}$ are in $[0, 1]$, we have $u_i(j, \mathcal{X}), u_i(j, \mathcal{Y}), u_i(\mathcal{X}), u_i(\mathcal{Y}) \in [0, 1]$ for all i, j, which in turn implies that we have both $\max_{i,j}(\mathcal{X}), \max_{i,j}(\mathcal{Y}) \in [0, 1]$.

Using these properties above, along with the trick, we can conclude

$$\begin{aligned}\left|x_{i,j} \sum_{k\in[\alpha]} \max_{i,k}(\mathcal{Y}) - y_{i,j} \sum_{k\in[\alpha]} \max_{i,k}(\mathcal{X})\right| &\leq \sum_{k\in[\alpha]} \left|x_{i,j} \cdot \max_{i,k}(\mathcal{Y}) - y_{i,j} \cdot \max_{i,k}(\mathcal{X})\right| \\ &\leq \sum_{k\in[\alpha]} \Big(|x_{i,j} - y_{i,j}| + |u_i(k, \mathcal{X}) - u_i(k, \mathcal{Y})| + |u_i(\mathcal{X}) - u_i(\mathcal{Y})|\Big).\end{aligned}$$

Similarly, we also have

$$\begin{aligned}&\left|\max_{i,j}(\mathcal{X}) \sum_{k\in[\alpha]} \max_{i,k}(\mathcal{Y}) - \max_{i,j}(\mathcal{Y}) \sum_{k\in[\alpha]} \max_{i,k}(\mathcal{X})\right| \\ &\qquad \leq \sum_{k\in[\alpha]} \Big(|u_i(j, \mathcal{X}) - u_i(j, \mathcal{Y})| + 2|u_i(\mathcal{X}) - u_i(\mathcal{Y})| + |u_i(k, \mathcal{X}) - u_i(k, \mathcal{Y})|\Big).\end{aligned}$$

Plugging all these back into (3.29), we have

$$\begin{aligned}\left|F_{i,j}(\mathcal{X}) - F_{i,j}(\mathcal{Y})\right| \leq (1+\alpha)\cdot|x_{i,j}-y_{i,j}| + (1+3\alpha)\cdot|u_i(\mathcal{X})-u_i(\mathcal{Y})| \\ + (1+\alpha)\cdot|u_i(j,\mathcal{X})-u_i(j,\mathcal{Y})| + 2\cdot\sum_{k\in[\alpha]}|u_i(k,\mathcal{X})-u_i(k,\mathcal{Y})|.\end{aligned}$$

Finally, we bound $|u_i(k,\mathcal{X}) - u_i(k,\mathcal{Y})|$ in terms of $\|\mathcal{X}-\mathcal{Y}\|_1$. Let S be the set of pure strategy profiles. Then, by applying the trick and the fact that all payoffs are in $[0,1]$, it follows that

$$|u_i(k,\mathcal{X}) - u_i(k,\mathcal{Y})| \leq \sum_{\boldsymbol{s}\in S_{-i}}\left|\prod_{q\neq i}x_{q,s_q} - \prod_{q\neq i}y_{q,s_q}\right| \leq \sum_{\boldsymbol{s}\in S_{-i}}\sum_{q\neq i}|x_{q,s_q}-y_{q,s_q}| \leq \alpha^{n-1}\|\mathcal{X}-\mathcal{Y}\|_1$$

$$|u_i(\mathcal{X}) - u_i(\mathcal{Y})| \leq \sum_{\boldsymbol{s}\in S}\left|\prod_{q\in[n]}x_{q,s_q} - \prod_{q\in[n]}y_{q,s_q}\right| \leq \sum_{\boldsymbol{s}\in S}\sum_{q\in[n]}|x_{q,s_q}-y_{q,s_q}| \leq \alpha^{n}\|\mathcal{X}-\mathcal{Y}\|_1.$$

Applying these inequalities, along with $|x_{i,j}-y_{i,j}| \leq \|\mathcal{X}-\mathcal{Y}\|_1$, we get

$$\left|F_{i,j}(\mathcal{X}) - F_{i,j}(\mathcal{Y})\right| \leq 10\alpha^{n+1}\cdot\|\mathcal{X}-\mathcal{Y}\|_1.$$

This finishes the proof Lemma 3.7.3.

3.8 Open Problems

Can the number of strategies be further reduced from seven in our PPAD-hardness result? Specifically, could we construct an anonymous game similar to the radix game $\mathcal{G}_{n,N}$, particularly its set of approximate Nash equilibria after perturbation, but without the four special (auxiliary) pure strategies $\{q_1, q_2, r_1, r_2\}$? While we believe this to be possible, constructing such a game can be highly non-trivial and would require specifying different payoffs for many of the possible outcomes seen by each player. Accordingly, proving a result similar to Lemma 3.4.3 after duplicating the first strategy would be even more difficult.

However, even the construction of such a game would only reduce the number of strategies used in the hardness proof down to three (due to the strategy duplication in the generalized radix game later), leading to the next open question: Is there an FPTAS for two-strategy anonymous games? As was posited by Daskalakis and Papadimitriou, it remains unclear whether a rational two-strategy anonymous game always has a rational Nash equilibrium. Additionally, in their sequence of paper's proving a PTAS for a bounded number of strategies, Daskalakis and Papadimitriou found that the form of the $\boldsymbol{Pr}_{\mathcal{X}}[p, \mathbf{k}]$ is significantly simpler for two-strategy anonymous games. Correspondingly, we found that constructing useful gadgets for reductions with just two strategies to be very difficult, suggesting that an FPTAS for two-strategy anonymous games is certainly a possibility.

Moreover, could there be an FPTAS for anonymous games with any bounded number of pure strategies? There is no clear way to strengthen our current construction to obtain a PPAD-hardness result for $1/\text{poly}(n)$-approximate Nash equilibrium. In order for the estimation lemma to hold, we need $x_{i,1} + x_{i,2} \approx \delta^i$ for all i. So even if we set $N = 2$, ensuring that $x_{i,1} + x_{i,2} = \delta^i \pm O(1/\text{poly}(n))$ would still not be sufficient for the estimation lemma to hold. Accordingly, in order to modify our construction to get such a hardness result, we would need to construct an anonymous game, which contains n players with the same properties as the main players in the generalized radix game, but with the additional property that $O(1/\text{poly}(n))$ shifts in the payoffs would only cause $O(1/2^{\text{poly}(n)})$ shifts in $x_{i,1} + x_{i,2}$, which seems incredibly unlikely.

CHAPTER 4

INDIVIDUAL SENSITIVITY PREPROCESSING FOR DATA PRIVACY

This was joint work with Rachel Cummings.

The sensitivity metric in differential privacy, which is informally defined as the largest marginal change in output between neighboring databases, is of substantial significance in determining the accuracy of private data analyses. Techniques for improving accuracy when the average sensitivity is much smaller than the worst-case sensitivity have been developed within the differential privacy literature, including tools such as smooth sensitivity, Sample-and-Aggregate, Propose-Test-Release, and Lipschitz extensions.

In this work, we provide a new and general Sensitivity-Preprocessing framework for reducing sensitivity, where efficient application gives state-of-the-art accuracy for privately outputting the important statistical metrics median, mean, and variance when no underlying assumptions are made about the database. In particular, our framework compares favorably to smooth sensitivity for privately outputting median, in terms of both running time and accuracy. Furthermore, because our framework is a preprocessing step, it can also be complementary to smooth sensitivity and any other private mechanism, where applying both can achieve further gains in accuracy.

We additionally introduce a new notion of *individual sensitivity* and show that it is an important metric in the variant definition of *personalized differential privacy*. We show that our algorithm can extend to this context and serve as a useful tool for this variant definition and its applications in markets for privacy.

Given the effectiveness of our framework in these important statistical metrics, we further investigate its properties and show that: (1) Our construction is conducive to efficient implementation with strong accuracy guarantees, evidenced by an $O(n)$ implementation for median (with presorted data), and $O(n^2)$ implementation for more complicated functions

such as mean, α-trimmed mean, and variance. (2) Our construction is both NP-hard and also optimal in the general setting (3) Our construction can be extended to higher dimensions, although it incurs accuracy loss that is linear in the dimension.

4.1 Introduction

Differentially private algorithms for data analysis guarantee that any individual entry in a database has only a bounded effect on the outcome of the analysis [161]. These algorithms ensure that the outcomes on any pair of neighboring databases—that differ in a single entry—are nearly indistinguishable. This is typically achieved by perturbing the analysis or its output, using noise that scales with the magnitude of change in the analysis between neighboring databases. This perturbation necessarily leads to decreased accuracy of the analysis. A fundamental challenge in differentially private algorithm design is to simultaneously satisfy privacy guarantees and provide accurate analysis of the database. Privacy alone can be achieved by outputting pure noise, but this fails to yield useful insights about the data. Intuitively, stronger privacy guarantees should yield weaker accuracy guarantees. Quantifying this privacy-accuracy tradeoff has been one major contribution of the existing differential privacy literature. In the last several years, accurate and differentially private algorithms have been designed for a diverse collection of data analysis tasks (see [162] for a survey), and have been implemented in practice by major organizations such as Apple, Google, Uber, and the U.S. Census Bureau. The formal guarantees of differential privacy give sharp contrast to ad hoc privacy measures such as anonymization and aggregation, which have both led to infamous privacy violations [163, 164].

We formalize data analysis tasks as functions that map from the space of all databases to real-valued outputs. The *global sensitivity* of a function is the worst-case difference in the function's value between all pairs of neighboring databases. Since differential privacy guarantees must hold for all pairs of neighboring databases, this is the scale of noise that must be added to preserve privacy. Strong bounds on global sensitivity imply that the function

is well-behaved over the entire data universe, and often allows for privacy-preserving output with strong accuracy guarantees. However, this worst-case measure allows a single outlier database to significantly skew the accuracy of the privacy-preserving algorithm for all databases. Although it is necessary to preserve the privacy of outlying databases, we would prefer to add less noise for improved accuracy guarantees when the average-case sensitivity is far smaller than the worst-case. A variety of well-known techniques have been employed to address this problem including smooth sensitivity and Sample-and-Aggregate [11], Propose-Test-Release [12], and Lipschitz extensions [13, 14, 15].

Initial work in this space considered a database-specific definition of sensitivity, known as *local sensitivity*, which is the maximum change in the function's value between a given database and its neighbors [11, 12]. Ideally, we would like to add noise that scales with the local sensitivity of each database. This would allow us to add less noise to well-behaved regions of the database universe, and only the outliers would require substantial noise. Unfortunately this procedure does not satisfy differential privacy because the amount of noise added to a given database may be highly disclosive. To avoid this information leakage, [11] defined an intermediate notion of *smooth sensitivity*, which smoothed the amount of noise added across databases to preserve differential privacy once again. This technique was also combined with random subset sampling to give an efficient and private procedure, Sample-and-Aggregate, with strong error guarantees when each database was well-approximated by a random subset of its entries [11].

Later work considered partitioning the database universe into well-behaved and outlying databases. Propose-Test-Release defined this partition with respect to local sensitivity and gave accurate outputs only on databases that were sufficiently far from outliers [12]. Propose-Test-Release avoided some of the information leakage issues by outputting NULL for any outlying database, and gave efficient implementations for a variety of important functions. The Lipschitz extension framework instead partitioned according to global sensitivity, by identifying a subset of the data universe where the given function had small global sensitivity.

On this subset, the function of interest is simply a Lipschitz function with the constant defined as the small global sensitivity [13, 14, 15]. Extending the Lipschitz function to the remaining data universe achieves a function with small global sensitivity that is identical to the original function on the well-behaved databases. Applying any differential privacy algorithm to this Lipschitz function will allow for the use of a much smaller global sensitivity input and will achieve high accuracy on the well-behaved databases.

In this work, we introduce a Sensitivity-Preprocessing framework that will similarly approximate a given function with a sensitivity bounded function, which we call the Sensitivity-Preprocessing Function. Our Sensitivity-Preprocessing Function will take advantage of the specific metric space structure of the data universe to give a more constructive approach. At a high-level, while Lipschitz extensions are initialized with a well-behaved subset of the data universe, our algorithm will find this well-behaved subset as it constructs the Sensitivity-Preprocessing Function. As a result, our procedure will be much more localized and can always give an exponential-time construction even in the most general setting, whereas Lipschitz extensions can often be uncomputable. In addition, similar to smooth sensitivity, we achieve optimality and NP-hardness guarantees for accuracy in this generalized setting under several reasonable metrics of optimality.

Furthermore, our Sensitivity-Preprocessing Function only requires a simple recursive construction that is more conducive to efficient implementation, which we achieve for important statistical functions such as median, mean, and variance. These functions have been of particular interest for similar techniques because they are highly important statistics and also have large worst-case sensitivity, but small average sensitivity. We will compare our results to previous results in the following section, where our framework gives state-of-the-art accuracy for median and mean when no underlying assumptions are made to the database. The key assumption that we would aim to avoid is that data points are drawn iid, which is a popular assumption in previous results for outputting functions such as mean (i.e., Propose-Test-Release [12]). While this assumption is quite standard, we contend that

real world data are often not iid, and so consideration of the more general case is still an important problem. Comparing our framework to those that apply the iid assumptions (and sometimes further assumptions, such as being drawn from a Gaussian [**KLSU18**]), we will also achieve similarly high accuracy for well concentrated databases, but concede that mechanisms specifically catered for that setting will often be superior. However, we note that because our framework is a preprocessing routine, it can be run before applying any differentially private mechanism such as Propose-Test-Release to achieve further gains in accuracy. In avoiding this iid assumption, the primary technique we will then compare our framework with will be smooth sensitivity which is popularly used for privately outputting median. Both frameworks have database-specific accuracy and direct comparison will be difficult, but we give strong evidence in the next section of why our framework compares favorably to smooth sensitivity for median.

The localized construction of our Sensitivity-Preprocessing Function will also allow us to tailor the new sensitivity parameters beyond previous techniques. To this end, we introduce a more refined sensitivity metric, which we call *individual sensitivity*, and show that it is important for a variant definition of *personalized differential privacy* introduced in [165], and used in subsequent works on market design for private data [166, 167]. We can apply our construction as a preprocessing step for more refined sensitivity tailoring to take advantage of personalized differential privacy guarantees. We believe this application of our results may of independent interest for future work in these directions.

In this work we cover a broad range of the more immediate results from this new framework, but believe that there is still a substantial amount of work in this direction. While some of our proofs will become involved, all of our results follow from first principles, suggesting the potential for further results using more sophisticated tools within this framework. These further results include: efficient implementations of more difficult functions such as linear regression; optimizing the trade-off between decreasing the sensitivity parameter and the error incurred by our Sensitivity-Preprocessing for specific functions; applying our algorithm

in the markets for privacy literature; and variants of our algorithm that are optimized for specific computational settings or application domains.

4.1.1 Differential Privacy and Sensitivity

We first give some of the basic definitions associated with differential privacy that will be useful for the remainder of the paper. The first definition is the notion of neighboring databases, where two databases are considered neighbors if the only difference is that the data of only one individual has been added or removed.

Definition 4.1.1. Given a set of databases $\mathcal{D}$, we say that two databases $D, D' \in \mathcal{D}$ are neighboring, or $d(D, D') = 1$, if the only difference is one individuals data has been added or removed. For much of this result, we will consider $\mathcal{D}$ to be the set of all real-valued vectors, and D, D' are neighbors if one has an additional entry in it's vector but are otherwise identical

The goal is then to protect the privacy of this one individual adding or removing their data by ensuring that the output from the mechanism maintains the privacy of this change by drawing from a similar probability distribution, hence *differential* privacy.

Definition 4.1.2 (Differential Privacy [161])**.** Given a set of databases, or data universe, $\mathcal{D}$ and some metric space $\mathcal{O}$. Let M be a randomized mechanism mapping $D \in \mathcal{D}$ to $s \in \mathcal{O}$. We say that this mechanism M is ϵ-differentially private if for any $D, D' \in \mathcal{D}$ such that $d(D, D') = 1$ and any subset $S \subset \mathcal{O}$, we must have

$$\mathbb{P}\left[M(D) \in S\right] \leq e^{\epsilon}\mathbb{P}\left[M(D') \in S\right]$$

Given that this considers preserving the privacy of all neighboring databases, it is then natural to consider the maximum change in the output of neighboring databases if our goal is to privately release a given function. More specifically, suppose we have some function, $f : \mathcal{D} \to \mathbb{R}$, mapping databases to the reals. For instance this could simply be the

function that computes the mean of each database. In order to output this function privately, it becomes critical to consider the maximal change that can occur between neighboring databases, defined as the global sensitivity.

Definition 4.1.3 (Global Sensitivity)**.** Given a function $f : \mathcal{D} \to \mathbb{R}$, we let $\Delta(f)$ denote the global sensitivity of f which is defined as

$$\Delta(f) \stackrel{\text{def}}{=} \max_{D,D':d(D,D')=1} |f(D) - f(D')|$$

4.1.2 Our Results

Our results will primarily revolve around the *Sensitivity-Preprocessing Function*, which we introduce below. It is an alternate schema for fitting a general function to a sensitivity-bounded function in the context of differential privacy. More specifically, we consider the general problem of taking any function $f : \mathcal{D} \to \mathbb{R}$ and constructing a new function $g : \mathcal{D} \to \mathbb{R}$ that satisfies given sensitivity parameters and minimizes the difference $|f(D) - g(D)|$ over all databases $D \in \mathcal{D}$. The sensitivity parameters we consider will be more refined and we define *individual sensitivity*, which is the maximum change in a function's value from adding or removing a single specific data entry. We use Δ_i to denote individual sensitivity to the i-th data entry. When a database is comprised of data from multiple individuals, Δ_i captures the sensitivity of the function to person i's data.

In this section, we first give an overview of our recursively constructed Sensitivity-Preprocessing Function that works in a highly generalized setting, along with the corresponding runtime and error guarantees. We then examine the optimality and hardness of this general function in the context of minimizing $|f(D) - g(D)|$ over all databases $D \in \mathcal{D}$. While constructing our Sensitivity-Preprocessing Function will require exponential time in general, we show that it can be simply and efficiently implemented in $O(n)$ time for median, and in $O(n^2)$ time for several other important statistical measures including mean and variance. We show that our Sensitivity-Preprocessing Function tailors an important

metric (individual sensitivity) in the variant definition of *personalized differential privacy*, which provides different privacy guarantees to different individuals in the same database, and is a useful tool in the design of markets for privacy. We further generalize our construction of Sensitivity-Preprocessing Function to bound the ℓ_1 sensitivity of 2-dimensional functions $f : \mathcal{D} \to \mathbb{R}^2$, and show that such techniques cannot be extended to higher dimensions.

Sensitivity-preprocessing function overview

Our construction of the Sensitivity-Preprocessing Function is similar to the Lipschitz extension framework, but we extend only from the empty set. We start with $f(\emptyset) = g(\emptyset)$, and inductively construct g for larger databases while trying to achieve two desiderata: (1) maintain the appropriate individual sensitivity bounds; and (2) keep g as close as possible to f. The first objective will be strictly maintained, and we will optimize over the second objective.

The primary difficulty in this construction is that we often consider $\mathcal{D}$ to be infinite. As a result, checking to make sure we do not violate any sensitivity constraints when defining g on a new database can require checking all databases on which g was previously defined. For example, general Lipschitz extensions require checking all previously defined databases to extend to another database, which can often be uncomputable for general functions. To avoid these uncomputability issues, we take advantage of the lattice structure of neighboring databases in the differential privacy landscape. This will allow us to give a far more localized construction that critically utilizes the following two key properties of the data universe metric space:

1. While each database could have infinitely many neighboring databases, it only has a finite number of neighbors with strictly fewer entries.

2. Any two neighbors of a strictly larger database must also be neighbors of a strictly smaller database.

These properties ensure that whenever we define g on a new database, we only need to check that sensitivity constraints of strictly smaller neighboring databases are satisfied. Once we have found the feasible range of g that does not violate any sensitivity constraints, we will define g to be as close as possible to f within this feasible range.

Definition 4.1.4 (Informal version of Definition 4.3.1)**.** Given a function $f : \mathcal{D} \to \mathbb{R}$ and fixed sensitivity parameters, we recursively define our Sensitivity-Preprocessing Function $g : \mathcal{D} \to \mathbb{R}$ such that $g(\emptyset) = f(\emptyset)$[1] and for any $D \in \mathcal{D}$,

$$g(D) = \text{closest point to } f(D) \text{ in } \textsc{Feasible}(D),$$

where $\textsc{Feasible}(D)$ is the set of all points that do not violate the sensitivity constraints based upon $g(D')$ for all neighbors D' of D with fewer entries.

The recursive structure of this function allows us to compute $g(D)$ by only looking at the subsets of D, which unfortunately takes exponential time. Later in the paper (Sections 4.5 and 4.6), we utilize the simplicity of the recursive structure to efficiently implement this algorithm for several functions of interest that exhibit additional structure. Theorem 4.1.5 summarizes our main result on the running time and accuracy guarantees of our Sensitivity-Preprocessing Function algorithm.

Theorem 4.1.5 (Informal version of Theorem 4.3.3)**.** *For any function $f : \mathcal{D} \to \mathbb{R}$ and desired sensitivity bounds $\{\Delta_i\}$, let $g : \mathcal{D} \to \mathbb{R}$ be the Sensitivity-Preprocessing Function of f. Given query access to f in $T(n)$ time for a database of size n, we give $O((T(n) + n)2^n)$ time access to $g(D)$ for any $D \in \mathcal{D}$ with n entries. We also give instance-specific bounds on each $|f(D) - g(D)|$ based on the sensitivity of f and $\{\Delta_i\}$.*

Our algorithm for computing the Sensitivity-Preprocessing Function g (Algorithm 10) is robust to informational assumptions. We only assume query access to f, and do not require any knowledge of the database universe $\mathcal{D}$ or the sensitivity of f.

[1] See Remark 4.3.2 for a discussion of how to initialize $g(\emptyset)$ if $f(\emptyset)$ is not well-defined.

This algorithm easily extends to functions that map to $\mathbb{R}^d$, by treating each dimension independently. See Remark 4.3.4 and Section 4.8 for more details on handling high-dimensional functions.

Approximate Optimality and Hardness

The construction of our Sensitivity-Preprocessing Function is quite simple in its greedy structure and requires exponential running time. To justify these two properties, we complement our algorithm with both optimality guarantees and hardness results.

In particular, we still consider the general problem of taking any function $f : \mathcal{D} \to \mathbb{R}$ and constructing a new function $g : \mathcal{D} \to \mathbb{R}$ with individual sensitivity bounds $\{\Delta_i\}$. The goal will then be to minimize the difference $|f(D) - g(D)|$ across databases $D \in \mathcal{D}$. Despite its simplicity, we show that our Sensitivity-Preprocessing Function still achieves a 2-approximation to the optimal function in the ℓ_∞ metric in this generalized setting.

Proposition 4.1.6 (Informal version of Corollary 4.4.4)**.** *Given any function $f : \mathcal{D} \to \mathbb{R}$ and sensitivity parameters $\{\Delta_i\}$, let $g : \mathcal{D} \to \mathbb{R}$ be our Sensitivity-Preprocessing Function. For any function $f^* : \mathcal{D} \to \mathbb{R}$ with individual sensitivity bounds $\{\Delta_i\}$,*

$$\max_{D \in \mathcal{D}} |f(D) - g(D)| \leq 2 \max_{D \in \mathcal{D}} |f(D) - f^*(D)| .$$

Our guarantees are even stronger because they also hold over finite subsets of the data universe. While Proposition 4.1.6 measures error in the worst-case over $\mathcal{D}$, we also show (Lemma 4.4.2) that when the optimal error is small on certain subsets of the data universe, then our error is also small.

Furthermore, we can show that our Sensitivity-Preprocessing Function is Pareto optimal: there is no strictly superior sensitivity-bounded function that improves accuracy over all databases.

Proposition 4.1.7 (Informal version of Lemma 4.4.5)**.** *Given any $f : \mathcal{D} \to \mathbb{R}$, let $g : \mathcal{D} \to \mathbb{R}$*

be the Sensitivity-Preprocessing Function of f with individual sensitivity parameters $\{\Delta_i\}$. For any $f^* : \mathcal{D} \to \mathbb{R}$ with individual sensitivity parameters $\{\Delta_i\}$, if there is some $D \in \mathcal{D}$ such that

$$|f(D) - f^*(D)| < |f(D) - g(D)|,$$

then there also exists some $D' \in \mathcal{D}$ such that

$$|f(D') - f^*(D')| > |f(D') - g(D')|.$$

These results imply that our Sensitivity-Preprocessing Function does quite well fitting to the original function under the metrics we are considering. However, it does take exponential time, so we complement these results by showing that getting the same approximation guarantees is NP-hard even for a single sensitivity parameter $\{\Delta_i\} = \Delta$.

Proposition 4.1.8 (Informal version of Proposition 4.4.6)**.** *Given any function $f : \mathcal{D} \to \mathbb{R}$ and sensitivity parameter Δ, it is NP-hard to construct any function $f^* : \mathcal{D} \to \mathbb{R}$ with sensitivity Δ that enjoys the same accuracy guarantees as our Sensitivity-Preprocessing Function.*

We further argue that it is uncomputable to do better than a 2-approximation in the ℓ_∞ metric, and also uncomputable to achieve even a constant approximation in any ℓ_p metric for $p < \infty$, which justifies our choice of metric. We believe that the combination of these results gives a strong indication that our Sensitivity-Preprocessing Function and corresponding exponential time construction is the best we can hope to achieve for the general problem under reasonable metrics of optimality.

Efficient Implementation for Important Statistical Measures

One of the main benefits of our Sensitivity-Preprocessing Function is that its simple recursive structure is conducive to giving simple efficient variants for specific functions through largely straightforward state space reductions and dynamic programming. To this end, we

give efficient implementations of our Sensitivity-Preprocessing Function for the important statistical functions mean, median, α-trimmed mean, maximum, minimum, and variance. These statistical metrics can be surprisingly difficult to release privately without assuming the input is restricted to some range, and often requires further assumptions for metrics like mean, such as data being drawn from identical and independent distributions. In fact, for Propose-Test-Release even the iid assumption is not sufficient to apply their framework to mean, and other works required further concentration properties such as being drawn from the normal distribution. However, our Sensitivity-Preprocessing Function does not require bounded sensitivity of the input function f, and can also avoid any iid requirements. As a result, we are able to efficiently implement each of these statistical metrics with no constraints on the inputs. It is important to note that our implementations only consider a single sensitivity parameter Δ, but we believe each can be efficiently extended for individual sensitivity parameters $\{\Delta_i\}$.

For each of these statistical metrics, we are simply implementing our Sensitivity-Preprocessing Function more efficiently, so all of the previously stated optimality guarantees still apply. To further strengthen these optimality guarantees, we give a more rigorous treatment of the error incurred by our efficient implementation of median, mean, and variance, which we consider to be three of the most fundamental statistical tasks.

Median We focus on privately and accurately computing median, because it has been extensively studied under smooth sensitivity [11]. Both our framework and smooth sensitivity provide database-specific accuracy guarantees, so a direct comparison of accuracy will be difficult. Nevertheless, we show that our framework compares favorably to smooth sensitivity on median.

We begin by stating our result. As in [11], we define $A^{(k)}(D) = \max_{0 \le t \le k+1}(x_{m+t} - x_{m+t-k-1})$ and $m = \frac{n+1}{2}$ [2], which is essentially the k-local sensitivity of median for database

[2]For simplicity we assume that n is odd and the median is x_m. The definition is nearly identical when n is even and will be treated more rigorously in Section 4.5.2

D. More formally,

$$A^{(k)}(D) = \max_{d(D,D')\leq k} LS_f(D').$$

We also need to define median on the empty set. Since this is not naturally defined, we allow it to be an input parameter $med(\emptyset)$ chosen by the data analyst as the estimated median. As our comparison will mostly be with smooth sensitivity which must assume values are in a bounded range $[min, max]$, the natural choice would be $med(\emptyset) = \frac{max-min}{2}$. Further, it would be natural in this setting to set our parameter $\Delta = \frac{max-min}{n}$.

Theorem 4.1.9. *Let $med : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ be the median function for the data universe of all finite-length real-valued vectors. For chosen parameters $med(\emptyset)$ and Δ, along with any database $D = (x_1, ..., x_n) \in \mathbb{R}^{<\mathbb{N}}$, if $x_1 \leq \cdots \leq x_n$ we give $O(n)$ time access to a function $g : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ with sensitivity Δ such that $g(D) = med(D)$ whenever $A^{(k)}(D) \leq 2(k+1)\Delta$ for $k \leq n/4$ and $med(D) \in [med(\emptyset) - \frac{n}{2}\Delta, med(\emptyset) + \frac{n}{2}\Delta]$.*

To interpret this accuracy result, we begin by comparing performance on the example considered by [11], where smooth sensitivity performed well. In fact, it was exactly this setting that motivated our choice of assumptions under which to show our framework outperforms smooth sensitivity. Consider an environment where data points $x_1, \ldots, x_n$ lie in a bounded range $[0, 1]$, and we naturally set $med(\emptyset) = 1/2$ and $\Delta = 1/n$. Consider the particular database $D = (x_1, \ldots, x_n)$, where $x_i = i/n$. In this example, it is easy to check that the assumptions are satisfied for our Sensitivity-Preprocessing Function to correctly output $g(D) = med(D)$. Privately answering the query $g(D)$ using the Laplace mechanism (see Definition 4.7.2 for a formal definition) outputs $med(D)$ plus noise with scale $\Delta/\epsilon = \frac{1}{\epsilon n}$. In contrast the noise parameter added under smooth sensitivity (without our sensitivity preprocessing) would have to scale $\frac{1}{\epsilon^2 n}$, which is asymptotically larger, and would thus yield significantly lower accuracy.

The assumptions in Theorem 4.1.9 that $A^{(k)}(D) \leq 2(k+1)\Delta$ for all $k \leq n/4$ and $med(D) \in [med(\emptyset) - \frac{n}{2}\Delta, med(\emptyset) + \frac{n}{2}\Delta]$ are then exactly the generalization of this

condition where our database still has values that are reasonably spread out, but also has low local sensitivity and we would still like to achieve high accuracy. Further note that these assumptions allow both the bottom and top quartile values to be arbitrarily small and large, implying that our construction is able to handle outliers well. Restricting our attention to databases that satisfy these conditions allows us to consider all of the databases under which smooth sensitivity performs well. Under these assumptions, smooth sensitivity will achieve a noise magnitude of $\frac{\Delta}{\epsilon^2}$, whereas we instead achieve an asymptotically better noise magnitude of $\frac{\Delta}{\epsilon}$. Note that smooth sensitivity requires a bounded range, which we are considering here to be of size $n\Delta$, giving a global sensitivity of $n\Delta$ for median. Accordingly, standard mechanisms would have noise magnitude of $\frac{n\Delta}{\epsilon}$, which is significantly worse than both our framework and smooth sensitivity.

It is important to acknowledge that our Sensitivity-Preprocessing framework will not outperform smooth sensitivity in general. For example, consider again the domain where all data points are bounded in $[0, 1]$, and consider the database D of all 1's. Then our $g(D) = 1$, and the Laplace Mechanism would have noise of magnitude $\frac{1}{\epsilon n}$. However, the smooth sensitivity of this database will be $e^{-\epsilon n/2}$, and the smooth sensitivity framework only requires noise of magnitude $e^{-\epsilon n/2}/\epsilon$. More generally, smooth sensitivity will often do better if most data entries are exponentially close to one value. To achieve benefits from both techniques, an analyst could simply apply the smooth sensitivity framework after our preprocessing step. Our preprocessing algorithm will only improve the smooth sensitivity parameters, so this approach will continue to achieve the strong accuracy guarantees of smooth sensitivity on highly concentrated databases. This can be done efficiently, as both smooth sensitivity and our preprocessing step take time $O(n^2)$ on database-ordered functions [3]. On the example above, if we first apply our Sensitivity-Preprocessing Function and then add noise based on the smooth sensitivity, then we will also have noise that scales

[3]Database-ordered functions will be defined in Section 4.5, and it will be seen that this general class can be implemented in $O(n^2)$ time for our framework. While it is outside the scope of this paper, it is straightforward to see that this also holds for the smooth sensitivity framework and that this property is preserved when applying our preprocessing to the median function. We leave formal proofs of these to future work.

approximately as $e^{-\epsilon n/2}/\epsilon$. Since our algorithm is a preprocessing step, it is compatible with all techniques for improving accuracy of differentially private algorithms. We view this as an exciting avenue for future work, to optimize the use of each tool under different parameter settings.

Mean. We first note that mean is not naturally defined on the empty set, so we define it to be an input parameter $\hat{\mu}$ chosen by the data analyst as the estimated mean. The analyst's choice of $\hat{\mu}$ should reflect her prior knowledge, and will play a role in our accuracy guarantees. Intuitively, if two databases have means that are exponentially far apart, we cannot hope to output both means accurately. As such, our Sensitivity-Preprocessing Function will be accurate on databases with mean reasonably close to $\hat{\mu}$. Our efficient implementation of mean will take $O(n^2)$ time and provide the following guarantees.

Theorem 4.1.10. *Let $\mu : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ be the mean function for the data universe of all finite-length real-valued vectors. For chosen parameters $\hat{\mu}$ and Δ, along with any database $D = (x_1, ..., x_n) \in \mathbb{R}^{<\mathbb{N}}$, we give $O(n^2)$ time access to a function $g : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ with sensitivity Δ such that,*

$$|g(D) - \mu(D)| \leq \max\left\{|\mu(D) - \hat{\mu}| - \frac{n}{3}\Delta, 0\right\} + \sum_{i=1}^{n} \max\left\{\frac{27\,|x_i - \mu(D)|}{n} - \Delta, 0\right\}.$$

Additionally, if we are guaranteed that each $x_i \in [\hat{\mu}+\alpha\Delta, \hat{\mu}+(\alpha+n)\Delta]$ for $\alpha \in [-n, 0]$, then $g(D) = \mu(D)$

As was previously mentioned, we claim that our framework gives state-of-the-art accuracy for privately outputting mean when no underlying assumptions are made on the database. It turns out that the lack of assumptions on the database makes outputting mean privately incredibly difficult, where even Propose-Test-Release was unable to privately output mean with iid assumptions. Often further assumptions such as data drawn from a normal distribution or other distributions that concentrate well are necessary to guarantee

highly-accurate private output of mean. To our knowledge, the best algorithm to output mean privately when no underlying assumptions are made is the naive algorithm, that simply considers the range $[\hat{\mu} - \frac{n}{2}\Delta, \hat{\mu} + \frac{n}{2}\Delta]$ of length $n\Delta$, and rounds up or down any value outside of this range. It is important to note that this range must be chosen independently of the database, as catering the range to the considered database can easily be shown to violate privacy. Restricting values to a range of $n\Delta$ will then ensure that global sensitivity is at most Δ and standard mechanisms can be applied from here. For all databases with values inside the range $[\hat{\mu} - \frac{n}{2}\Delta, \hat{\mu} + \frac{n}{2}\Delta]$ this will then give accurate output.

Note that our second accuracy guarantee similarly considers databases under which our preprocessing correctly outputs the mean. It can be immediately seen that the allowable range for values in the database extends beyond the range for the naive algorithm, and can in some ways be seen to double this range. Essentially, the minimum and maximum values must still be within $n\Delta$ for us to guarantee correctly outputting the mean, but the range under with minimum and maximum values can fall is now doubled. Given the significance of mean as a statistical metric, we still believe this improvement is of significance and is the first to improve upon the naive algorithm when no underlying assumptions are made with regard to the database.

To complement this comparison to the naive algorithm, we also give strong accuracy guarantees for all databases not just the ones output correctly in our preprocessing. Unpacking the bound in Theorem 4.1.10, the first term says that the mean of the database cannot be too far from $\hat{\mu}$. The second term considers the individual sensitivity of each data point, where $|x_i - \mu(D)|/n$ is roughly the amount the mean changes from adding x_i to the database. The sensitivity bound on g requires that each individual change can only be offset by an additive Δ, and we need to consider this contribution from each input. Intuitively, our error is small for databases whose mean is reasonably close to $\hat{\mu}$ and do not have many significant outliers, which is exactly what one would expect.

Variance. Variance is also not naturally defined on the empty set, so we define it to be 0 for simplicity. Our efficient implementation of variance takes $O(n^2)$ time and provides the following guarantee.

Theorem 4.1.11. *Let* ***Var*** $: \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ *be the variance function for the data universe of all finite-length real-valued vectors. For fixed parameter* Δ*, along with any database* $D = (x_1, ..., x_n) \in \mathbb{R}^{<\mathbb{N}}$*, we have* $O(n^2)$ *time access to a function* $g : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ *with sensitivity* Δ *such that,*

$$|g(D) - \textbf{Var}\,[D]\,| \leq \max\left\{\textbf{Var}\,[D] - \frac{n}{2}\Delta, 0\right\} + \sum_{i=1}^{n} \max\left\{\sum_{j=1}^{n} \frac{4(x_i - x_j)^2}{n^2} - \Delta, 0\right\}.$$

The primary takeaway from the bound in Theorem 4.1.11 is that databases with reasonably small variance and no major outliers will have low error bounds. The first term in the error bound says that the variance of our database cannot be too large, which follows from our choice of the empty set to be defined at 0. The second term is a bit more messy, but has a natural interpretation under the known deformulation of variance, where we can consider $\sum_{j=1}^{n}(x_i - x_j)^2/n^2$ to be the contribution of input x_i to the variance. This contribution can then be offset by Δ, and we need to consider this contribution from each input.

Personalized Privacy

Due to the preprocessing aspect of our Sensitivity-Preprocessing Function, we can also apply our framework to variant definitions of differential privacy. In particular, we consider *personalized differential privacy* introduced in [165], which allows for a more refined definition whereby each individual receives their own ϵ_i privacy parameter. Our definition of *individual sensitivity* is then motivated by this privacy variant, as it can be exactly seen as the complementary sensitivity measure for this variant. More specifically, most privacy mechanisms add noise proportional to $\Delta f/\epsilon$ for outputting functions $f : \mathcal{D} \to \mathcal{R}$ while still preserving ϵ-differential privacy. This intimate connection between Δf and

ϵ in the output accuracy will be equivalent for the individual sensitivity measures $\Delta_i(f)$ and its respective ϵ_i. Consequently, the necessary noise for personalized privacy will be proportional to $\max_i \Delta_i(f)/\epsilon_i$. We will formally prove this fact for two of the most fundamental mechanisms, Laplace and Exponential, and further remark (Remark 4.7.6) that this approach extends to any ϵ-differentially private mechanism.

Theorem 4.1.12 (Informal version of Propositions 4.7.3 and 4.7.5)**.** *For both the Laplace and Exponential Mechanisms, instead of adding noise proportional to Δ/ϵ, the added noise can be proportional to $\max_i \Delta_i/\epsilon_i$ to ensure personalized differential privacy for privacy parameters $\{\epsilon_i\}$.*

As a result, it is no longer necessarily optimal to set the individual sensitivity parameters $\{\Delta_i\}$ in our Sensitivity-Preprocessing Function to be equal, but instead set them according to the given $\{\epsilon_i\}$ privacy parameters towards the goal of having each $\Delta_i(g)/\epsilon_i$ roughly equal. This extends the interest in our Sensitivity-Preprocessing Function beyond the context of dealing with worst-case sensitivity being much greater than average sensitivity.

For example, consider a well-behaved function where $\{\Delta_i\} = \Delta$, and $\{\epsilon_i\} = \epsilon$ for all i except for some individual j where $\epsilon_j = \epsilon/2$. Under this situation it may instead be optimal to halve the individual sensitivity of j, which will allow adding half as much noise while only incurring a small additive error by restricting the sensitivity of just one person.

In addition, the error bounds from our general procedure allow for the intuitive fact that increasing any individual sensitivity will increase the accuracy of our preprocessing step for databases including that individual. We note that when trying to preserve the fraction $\Delta_i(g)/\epsilon_i$, any increase in ϵ_i (reduced privacy for individual i) will allow us to increase our Δ_i parameter, improving accuracy as desired. In this way, our Sensitivity-Preprocessing Function is able to fully take advantage of the heterogeneous ϵ_i in the variant definition of personalized privacy, and is the first to give accuracy bounds that increase/decrease independently with respect to each ϵ_i.

We believe that these accuracy guarantees can be of further interest in the context of

markets for privacy, where individuals sell their data to an analyst and demand different amounts of privacy, represented by their respective ϵ_i. The trade-off between privacy and accuracy is naturally formalized in these markets through the analyst's budget for procuring accurate estimates of population statistics. Applying our Sensitivity-Preprocessing Function to achieve individualized privacy guarantees will allow the analyst to more optimally balance these trade-offs because the accuracy will respond proportionally to changes in privacy for each individual.

Higher-Dimensional Extensions for ℓ_1 Sensitivity

Our Sensitivity-Preprocessing Function was only defined for 1-dimensional $f : \mathcal{D} \to \mathbb{R}$. We also consider the setting where $f : \mathcal{D} \to \mathbb{R}^d$. We note that our Sensitivity-Preprocessing Function could instead be given parameters $\{\Delta_i\}$ where $\Delta_i = (\Delta_{i,1}, ..., \Delta_{i,d})$ has different sensitivity parameters for each dimension of the function. We could then apply our Sensitivity-Preprocessing Function to each dimension independently and would achieve the corresponding bounds on sensitivity. However, this approach would require adding noise independently to each dimension when applying a differentially private mechanism on the Sensitivity-Preprocessing Function. Instead, we would like to only require bounds on the ℓ_1 sensitivity of our constructed function.

We give a natural extension of our Sensitivity-Preprocessing Function to higher dimensions, and show that the accuracy guarantees continue to hold in ℓ_1-distance when f is 2-dimensional (Theorem 4.8.3). We also show that this construction fails to extend to higher dimensions because a key fact about the intersection of ℓ_1 balls only holds in 1 and 2 dimensions.

4.1.3 Related Work

Our work touches upon several areas of interest. We first discuss previous work on dealing with outlying databases within the data universe, and then discuss previous work on

personalized differential privacy and its use within the markets for privacy literature.

Worst-case vs average-case sensitivity

Instance-specific noise for dealing with worst-case sensitivity was first introduced in [11], where they considered adding noise proportional to *local sensitivity*. In order to avoid leaking too much information through noise added by local sensitivity, [11] constructed a *smooth sensitivity* metric that minimized the instance-specific noise while still ensuring differential privacy. They further showed that smooth sensitivity could be efficiently computed and utilized for a variety of important functions for which average sensitivity was much smaller than *global sensitivity*. However, for some functions computing smooth sensitivity was NP-hard or even uncomputable, which inspired the introduction of Sample-and-Aggregate, a technique that preserved privacy and was efficient on all functions with bounded range and for sufficiently large databases. The general idea was to approximate the function with random subsets of the given database in order to impose stronger bounds on the sensitivity of this approximation. Combining this with smooth sensitivity allowed for strong error guarantees under the assumption that random subsets of the database often well-approximated the full database.

In order to avoid some of these assumptions, an alternate framework, Propose-Test-Release, was provided in [12], which also heavily relied on the notion of local sensitivity. In particular, their framework would check if a given database was "far away" from an outlier, and only release an accurate estimate of the output under this specific circumstance, while outputting $Null$ otherwise. Furthermore, this algorithm would define the outlying databases by explicitly setting the allowable upper bound on local sensitivity. They show how to implement this framework efficiently for several important functions, and give strong error guarantees when the mechanism does not output $Null$.

Both of these frameworks relied upon local sensitivity, which is still a worst-case metric. It is possible for most databases to have high local sensitivity while still having

small average sensitivity. To remedy this issue, previous work instead considered fitting the original function to one with global sensitivity closer to the average sensitivity. This preprocessing step can largely be thought of as forcing the output of outlying databases to be closer to that of the well-behaved databases. This procedure then fits in the general notion of Lipschitz extensions. Informally, Lipschitz extensions show that there always exists an extension of a smooth function restricted to a subspace to the entire metric space. By considering "smoothness" in the context of differential privacy to be the sensitivity of the function, previous work generally considers the restricted subspace to be the well-behaved databases.

Lipschitz extensions were first implicitly used in [13] under the context of *node* differential privacy. This work considered restricting the maximum degree of graphs for outputting a variety of graph statistics in bounded-degree graphs. This work was then extended in [15] which gave efficient Lipschitz extensions for higher-dimensional functions on graphs such as degree distribution. In this work, [15] further utilize Lipschitz extensions for a generalization of the exponential mechanism. Lipschitz extensions were also considered in [14] were the goal was to achieve a *restricted sensitivity* under a certain hypothesis of the database universe and extending to the entire data universe with this global sensitivity constraint. While this procedure was in general computationally inefficient, [14] gave efficient versions for subgraph counting queries and local profile queries.

Our technique of considering only strictly smaller neighboring databases is related to a technique used to achieve differential privacy over graphs. The *down sensitivity* [**RS15**] (also called *empirical global sensitivity* in [**CZ13**]) of a function at a graph G is the global sensitivity of the function when restricted to the space of all subgraphs of G. That is, it is the maximum change in the function's value between any two neighboring subgraphs of G. Similar to our work, this requires checking sensitivity on a smaller number of neighboring databases, and can allow less noise to be added to analysis on databases with small down sensitivity. Through this lens, our construction of Sensitivity-Preprocessing

Function can be viewed as ensuring that all databases have low down sensitivity. However, an important distinction between these two results is that down sensitivity considers all pairs of neighboring subgraphs of G, which, for example, may be the empty graph and a single node for a large graph G. To contrast, at each recursive step of our algorithm, we only consider only smaller neighbors of the current database, i.e., with one entry removed. This refined analysis means that a database might have large down sensitivity, and our Sensitivity-Preprocessing Function can still be accurate

Personalized privacy and markets for privacy

We show how our Sensitivity-Preprocessing framework can be applied to *personalized differential privacy*, where each user in the database has her own privacy parameter ϵ_i. This definition was first introduced by [165], in the context of purchasing data from privacy-sensitive individuals. A subsequent line of work on market design for private data [166, 168, 169, 170, 171, 165, 172, 167, 173, 174, 175] leveraged personalized privacy guarantees to purchase data with different privacy guarantees from individuals with heterogeneous privacy preferences. The vast majority of this work focused on the market design problem of procuring data, and not on the differentially private algorithms that provided personalized privacy guarantees. [167] gave a technique for achieving personalized privacy for linear functions by reweighting each person's data inversely proportional to their privacy guarantee. Unfortunately, this reweighting technique does not extend beyond linear functions. [166] proposed an even stronger notion of personalized privacy, that was both personalized and data-dependent, but did not give any algorithmic techniques to satisfy this definition.

Several other results gave mechanisms specific to personalized differential privacy by randomly keeping each individuals data in the database with probability proportional to their respective ϵ_i [176, 177, 178]. However, they are unable to provide corresponding error guarantees with such procedures for general functions. [179] gave a technique for providing two-tiered personalized privacy guarantees. Some users received differential privacy and

some users received a stronger guarantees of *local differential privacy*, where the users do not trust the data analyst to see their true data.

Finally, there is a small body work on high probability privacy guarantees and average-case privacy guarantees [180, 181, 182]. This work addresses a very different problem than we study here. These papers assume that databases are sampled according to some distribution over the data universe, and provide high probability guarantees with respect to the sampling distribution, allowing a failure of either privacy or accuracy on some set of unlikely databases. To contrast, we assume that databases are fixed, not randomly sampled. We provide privacy and accuracy guarantees that depend on the well-behavedness of a given function over the data universe, and our guarantees hold everywhere in the data universe.

4.1.4 Organization

In Section 4.2, we introduce some of the notation and basic definitions that will be used throughout the paper. In Section 4.3, we introduce our Sensitivity-Preprocessing Function and prove its general accuracy guarantees. In Section 4.4, we give optimality and hardness guarantees for our sensitivity-preprocessing procedure. In Section 4.5, we show that several important functions can be efficiently implemented in our framework, such as mean, median, maximum, minimum, and we also give strong error guarantees on the implementation of mean. In Section 4.6, we efficiently implement our framework for variance and give corresponding error guarantees. In Section 4.7, we prove several useful facts regarding individual sensitivity and the variant definition of personalized differential privacy, and show how our framework can be very useful in this context. In Section 4.8, we consider a natural extension of our algorithm that bounds a function's sensitivity in the ℓ_1 metric for 2 dimensions.

4.2 Preliminaries

We introduce the standard notion of differential privacy and the corresponding global sensitivity metric. We say that two databases are *neighboring* if they differ in at most one entry.

Definition 4.2.1 (Differential privacy [161])**.** A mechanism $\mathcal{M} : \mathcal{D} \to \mathcal{R}$ is *ϵ-differentially private* if for every pair of neighboring databases $D, D' \in \mathcal{D}$, and for every subset of possible outputs $\mathcal{S} \subseteq \mathcal{R}$,

$$\boldsymbol{Pr}[\mathcal{M}(D) \in \mathcal{S}] \leq \exp(\epsilon)\boldsymbol{Pr}[\mathcal{M}(D') \in \mathcal{S}].$$

Definition 4.2.2 (Global Sensitivity)**.** The *global sensitivity* of a function $f : \mathcal{D} \to \mathbb{R}^d$ is:

$$\Delta f = \max_{D,D',\, neighbors} \|f(D) - f(D')\|_1 .$$

Our result is primarily concerned with tailoring a more refined version of global sensitivity, for which we will need more specific notation for neighboring databases. In particular, we will consider the data universe $\mathcal{D}$ to be composed of (a possibly infinite) collection of individuals, where x_i will denote the data of individual i. Any database $D \in \mathcal{D}$ is then composed of the data of some finite subset of individuals I, so $D = \{x_i : i \in I\}$. For ease of notation, we will often assume that $D = (x_1, ..., x_n)$. Further, if individual i's data is contained in database D, then we will consider $D - x_i$ to be the database with individual i's data removed. Similarly, if i's data is not included in database D, then we consider the database $D + x_i$ to be the database with i's data added. We will also sometimes use $i \in D$ to denote that individual i's data is included in database D. Finally, we also assume that for any $D \in \mathcal{D}$, if $D' \subset D$ is non-empty, then $D' \in \mathcal{D}$.

With this notation, we introduce the notion of individual sensitivity that is the maximum change in output that is possible by adding individual i's data.

Definition 4.2.3 (Individual Sensitivity)**.** The *individual sensitivity* of a function $f : \mathcal{D} \to \mathbb{R}^d$ with respect to i is:

$$\Delta_i(f) \stackrel{\text{def}}{=} \max_{x_i, \{D : i \notin D\}} \|f(D) - f(D + x_i)\|_1 .$$

We further let $\{\Delta_i(f)\}$ denote the individual sensitivities of f to all individuals.

For reference, we also provide the definition of local sensitivity that will not be used in this work, but was referred to extensively in related works.

Definition 4.2.4 (Local Sensitivity)**.** The *local sensitivity* of a function $f : \mathcal{D} \to \mathbb{R}^d$ at database $D \in \mathcal{D}$ is:

$$\Delta_D f = \max_{D' \text{: neighbor of } D} \|f(D) - f(D')\|_1 .$$

4.3 Sensitivity-Preprocessing Function

In this section we formally define our Sensitivity-Preprocessing Function, give the corresponding constructive algorithm for accessing this function, and prove instance-specific error bounds between the original function and our Sensitivity-Preprocessing Function. Recall that our primary goal is to give an alternate schema for fitting a general function to a sensitivity bounded function. More specifically, suppose we are given a function $f : \mathcal{D} \to \mathbb{R}$ and desired sensitivity parameters $\{\Delta_i\}$, and want to produce another function $g : \mathcal{D} \to \mathbb{R}$ that closely approximates f, and has individual sensitivity at most Δ_i for all i.

The Sensitivity-Preprocessing Function will ultimately be defined as a simple greedy recursion that builds up from the empty set. The key insight is that we can take advantage of the particular metric space structure of databases such that defining our function on a new database only depends on the subsets of that database. We first use the fact that while each database could have infinitely many neighboring databases, it only has a finite amount of neighbors with strictly fewer entries. This will allow us to only consider the constraints incurred by each $D - x_i$ for some database D. In particular, for each $g(D - x_i)$ it is allowable

to place $g(D)$ anywhere in the region $[g(D - x_i) - \Delta_i, g(D - x_i) + \Delta_i]$. Intersecting each of these intervals will give the feasible region for $g(D)$, and we will greedily chose the point closest to $f(D)$. We then use the fact that any two neighbors of a strictly larger database must also be neighbors of a strictly smaller database. This will ensure that the intersection of all feasible intervals is non-empty, even under our greedy construction.

As a result, the Sensitivity-Preprocessing Function g is defined inductively starting from the empty set, and new data points are added one by one. The algorithm ensures that the value of g changes by at most Δ_i when new data point x_i is added, while minimizing the distance $|f(D) - g(D)|$ at every point.

Definition 4.3.1 (Sensitivity-Preprocessing Function)**.** Given any function $f : \mathcal{D} \to \mathbb{R}$ and non-negative parameters $\{\Delta_i\}$, we say that a function $g : \mathcal{D} \to \mathbb{R}$ is a Sensitivity-Preprocessing Function of f with parameters $\{\Delta_i\}$, if $g(\emptyset) = f(\emptyset)$[4] and

$$g(D) = \begin{cases} \textsc{Upper}(D), & \text{if } \textsc{Upper}(D) \leq f(D) \\ \textsc{Lower}(D), & \text{if } \textsc{Lower}(D) \geq f(D) \\ f(D), & \text{otherwise} \end{cases}$$

where $\textsc{Upper}(D) = \min_{j \in D}\{g(D - x_j) + \Delta_j\}$ and $\textsc{Lower}(D) = \max_{j \in D}\{g(D - x_j) - \Delta_j\}$.

If $\{\Delta_i\} = \Delta$ for some non-negative Δ, then we say that g is a Sensitivity-Preprocessing Function of f with parameter Δ.

The generalization from global sensitivity to individual sensitivities is critical to our personalized privacy results in Section 4.7. This generalization does not increase the running time, and it is easy to see that this yields global sensitivity equal to the maximum individual sensitivity.

[4]See Remark 4.3.2 for a discussion of how to initialize $g(\emptyset)$ if $f(\emptyset)$ is not well-defined.

4.3.1 Algorithmic Construction of Sensitivity-Preprocessing Function

The algorithm PREPROCESSING is presented in Algorithm 10. It begins by initializing $g(\emptyset) = f(\emptyset)$, and building up g to be defined on databases of increasing size. At each step, the algorithm ensures that no sensitivity constraints are violated and chooses the best value for $g(D)$ subject to those constraints.

For a given database D, UPPER(D) is the maximum value $g(D)$ can take without letting it increase too much from a smaller database (violating an individual sensitivity parameter). Similarly, LOWER(D) is the minimum value we can make $g(D)$ without letting it decrease too much from a smaller database. We then define $g(D)$ to be the value in $[\text{LOWER}(D), \text{UPPER}(D)]$ that is the closest to $f(D)$.

Algorithm 10: Sensitivity-Preprocessing Function Algorithm : PREPROCESSING($f : \mathcal{D} \to \mathbb{R}, \{\Delta_i\}, D$)

Input: Function $f : \mathcal{D} \to \mathbb{R}$, individual sensitivity bounds $\{\Delta_i\}$, and database D of size n.

Output: $g(D)$, where g satisfies individual sensitivity Δ_i for all i.

Initialize $g(\emptyset) = f(\emptyset)$ **for** *k=1, …, n* **do**

  for *every database* $D' \subseteq D$ *of size* k **do**

    Set $\text{UPPER}(D') = \min_{i \in D'} \{g(D' - x_i) + \Delta_i\}$

    Set $\text{LOWER}(D') = \max_{i \in D'} \{g(D' - x_i) - \Delta_i\}$

    Set $g(D') = \begin{cases} \text{UPPER}(D'), & \text{if } \text{UPPER}(D') \leq f(D') \\ \text{LOWER}(D'), & \text{if } \text{LOWER}(D') \geq f(D') \\ f(D'), & \text{otherwise} \end{cases}$

Output $g(D)$

This construction of g ensures that the individual sensitivity of g does not exceed Δ_i for each i. We can then use these bounds on the sensitivity of g to calibrate the scale of noise that must be added to ensure differential privacy. In the special case that $\Delta_i = \Delta$ for all i, then the global sensitivity of g is Δ, and we can add noise that scales with $O(\frac{\Delta}{\epsilon})$ to achieve ϵ-differential privacy. Note that this guarantee holds even if f has unbounded sensitivity. In Section 4.7, we show how to satisfy differential privacy under heterogeneous Δ_i.

Remark 4.3.2. Our algorithm is initialized using $f(\emptyset)$, and thus centers g around this point. In the case that $f(\emptyset)$ is undefined—for example, when f computes the mean of a database—the analyst should initialize $g(\emptyset)$ using some domain knowledge or prior beliefs on reasonable centering of the function. If no prior knowledge is available, the analyst can sample multiple databases and evaluate f on the samples to estimate a reasonable centering point for $g(\emptyset)$. The sensitivity bounds will still hold regardless of the centering of g, but accuracy may suffer if $g(\emptyset)$ is set to be far from most values of f.

In addition to sensitivity guarantees and runtime analysis, we also provide an instance-specific error bound. Unfortunately this bound will not be in a clean form, but it does capture the intuitive fact that if we increase any Δ_i then it is likely that accuracy also increases.

However, we are able to obtain a bit more intuition on our instance-specific error bounds, and can consider them in a similar context to local sensitivity. Given that our Sensitivity-Preprocessing Function defines a database recursively in terms of its subsets, it makes sense that our error guarantees will be in terms of these subsets. These error bounds can then be seen as capturing the sensitivity between the neighboring subsets of D. Analogously to local sensitivity, we will have larger errors for databases with high sensitivity between the neighboring subsets.

Theorem 4.3.3. *Given $T(n)$ time query access to an arbitrary function $f : \mathcal{D} \to \mathbb{R}$, and sensitivity parameters $\{\Delta_i\}$,* PREPROCESSING *provides $O((T(n) + n)2^n)$ time access to the Sensitivity-Preprocessing Function $g : \mathcal{D} \to \mathbb{R}$ such that $\Delta_i(g) \leq \Delta_i$ for all i. Further, for any database $D = (x_1, ..., x_n)$,*

$$|f(D) - g(D)| \leq \max_{\sigma \in \sigma_D} \sum_{i=1}^{|D|} \max\{\left|f(D_{\sigma(<i)} + x_{\sigma(i)}) - f(D_{\sigma(<i)})\right| - \Delta_{\sigma(i)}, 0\},$$

where σ_D is the set of all permutations on $[n]$, and $D_{\sigma(<i)} = (x_{\sigma(1)}, ..., x_{\sigma(i-1)})$ is the subset of D that includes all individual data in the permutation before the ith entry.

Remark 4.3.4. We can easily extend this theorem to $f : \mathcal{D} \to \mathbb{R}^d$ by running PREPROCESSING on each dimension independently in terms of sensitivity parameters and error bounds Specifically, suppose we were instead given parameters $\{\Delta_i\}$ where $\Delta_i = (\Delta_{i,1}, ..., \Delta_{i,d})$ has different sensitivity parameters for each dimension of the function. We could then consider the function restricted to a single dimension d', and run PREPROCESSING on this projection with sensitivity parameters $\{\Delta_{i,d'}\}$. This will give the desired sensitivity bounds in that single dimension, then running PREPROCESSING on all dimensions and composing across dimensions will give the appropriate Sensitivity-Preprocessing Function in d dimensions. In Section 4.8, we consider extensions to higher dimensions where each dimension is not treated independently.

4.3.2 Sensitivity-Preprocessing Function Correctness

We first prove that the Sensitivity-Preprocessing Function given in Definition 4.3.1 both meets the individual sensitivity criteria and is also defined on all databases.

Lemma 4.3.5. *For any function $f : \mathcal{D} \to \mathbb{R}$ and non-negative sensitivity parameters $\{\Delta_i\}$, if $g : \mathcal{D} \to \mathbb{R}$ is defined according to Definition 4.3.1, then g is defined on all databases $D \in \mathcal{D}$ and $\Delta_i(g) \leq \Delta_i$ for all i.*

Proof. It suffices to show that for any $D \in \mathcal{D}$ with at least one entry and for any $x_i \in D$, we have $g(D - x_i) - \Delta_i \leq g(D) \leq g(D - x_i) + \Delta_i$. By our construction, this must always be true if $\text{LOWER}(D) \leq g(D) \leq \text{UPPER}(D)$ for any $D \in \mathcal{D} \setminus \{\emptyset\}$. Our construction of g will always place $g(D) \in [\text{LOWER}(D), \text{UPPER}(D)]$ if the interval is non-empty, so it suffices to show that for all $D \in \mathcal{D} \setminus \{\emptyset\}$,

$$\text{LOWER}(D) \leq \text{UPPER}(D).$$

We will prove this by induction starting with $D = x_i$ with one entry. Therefore, $\text{UPPER}(D) = f(\emptyset) + \Delta_i$ and $\text{LOWER}(D) = f(\emptyset) - \Delta_i$, which implies our desired inequality

because $\Delta_i \geq 0$.

We now consider an arbitrary D and assume that our claim holds for all $D' \subset D$. Let $x_k \in D$ minimize $g(D - x_i) + \Delta_i$ over all $x_i \in D$, so

$$\textsc{Upper}(D) = g(D - x_k) + \Delta_k,$$

and let $x_j \in D$ maximize $g(D - x_i) - \Delta_i$ over all $x_i \in D$, so

$$\textsc{Lower}(D) = g(D - x_j) - \Delta_j.$$

If $k = j$ then the desired inequality immediately follows. Otherwise we consider $D - x_k - x_j$. By our inductive hypothesis, we know $\textsc{Lower}(D - x_k) \leq g(D - x_k) \leq \textsc{Upper}(D - x_k)$, so

$$g(D - x_k) \geq \textsc{Lower}(D - x_k) \geq g(D - x_k - x_j) - \Delta_j.$$

Similarly, we have $\textsc{Lower}(D - x_j) \leq g(D - x_j) \leq \textsc{Upper}(D - x_j)$, so

$$g(D - x_j) \leq \textsc{Upper}(D - x_j) \leq g(D - x_k - x_j) + \Delta_k.$$

Combining these inequalities gives $g(D - x_k) + \Delta_j \geq g(D - x_j) - \Delta_k$, which implies our desired result.

□

4.3.3 Error Bounds for Sensitivity-Preprocessing Function

We now prove the desired instance-specific error bounds between the original function and our Sensitivity-Preprocessing Function.

Lemma 4.3.6. *For any function $f : \mathcal{D} \to \mathbb{R}$ and non-negative sensitivity parameters $\{\Delta_i\}$,*

if $g : \mathcal{D} \to \mathbb{R}$ is defined according to Definition 4.3.1, then for any database $D \in \mathcal{D}$,

$$|f(D) - g(D)| \leq \max_{\sigma \in \sigma_D} \sum_{i=1}^{|D|} \max\{\left|f(D_{\sigma(<i)} + x_{\sigma(i)}) - f(D_{\sigma(<i)})\right| - \Delta_{\sigma(i)}, 0\},$$

where σ_D is the set of all permutations on $[n]$, and $D_{\sigma(<i)} = (x_{\sigma(1)}, \ldots, x_{\sigma(i-1)})$ is the subset of D that includes all individual data in the permutation before the ith entry.

Proof. We will prove this claim inductively and first consider $D = x_j$ with one entry for some j. We need to show

$$|f(D) - g(D)| \leq \max\{|f(D) - f(\emptyset)| - \Delta_j, 0\},$$

which follows easily from construction of g. We now consider an arbitrary D and assume that the claim is true for all $D' \subset D$. From our construction we claim that

$$|f(D) - g(D)| \leq \max_{x_i \in D}\{|f(D) - g(D - x_i)| - \Delta_i, 0\}.$$

This follows from the fact that if $f(D) = g(D)$ then we must have $|f(D) - g(D - x_i)| \leq \Delta_i$ for all i, and otherwise there must be some $x_i \in D$ such that the constraint on $g(D)$ with respect to Δ_i is tight. Using this fact we can bound $|f(D) - g(D)|$ in the following way:

$$\begin{aligned}
|f(D) - g(D)| &\leq \max_{x_i \in D}\{|f(D) - g(D - x_i)| - \Delta_i, 0\} \\
&= \max_{x_i \in D}\{|f(D) - f(D - x_i) + f(D - x_i) - g(D - x_i)| - \Delta_i, 0\} \\
&\leq \max_{x_i \in D}\{|f(D) - f(D - x_i)| - \Delta_i + |f(D - x_i) - g(D - x_i)|, 0\} \\
&\leq \max_{x_i \in D}\{\max\{|f(D) - f(D - x_i)| - \Delta_i, 0\} + |f(D - x_i) - g(D - x_i)|\}
\end{aligned}$$

We then apply the inductive hypothesis to $|f(D - x_i) - g(D - x_i)|$, which immediately implies our desired bound. □

4.3.4 Proof of Theorem 4.3.3

Proof of Theorem 4.3.3. The individual sensitivity guarantees are given by Lemma 4.3.5, and the error bounds are given by Lemma 4.3.6. It then remains to show the running time. If we assume $T(n)$ time access to f for a database with n entries, then because we need to query each subset of D, this will contribute time $O(T(n)2^n)$. Furthermore, for each subset we need to compute $\textsc{Upper}(D)$ and $\textsc{Lower}(D)$ which takes $O(n)$ time for each subset. This then gives our full runtime of $O((T(n) + n)2^n)$.

□

4.4 Optimality and Hardness of Sensitivity-Preprocessing Function

Our algorithm in Section 4.3 took exponential time to query the Sensitivity-Preprocessing Function g at each database D of interest, and, while we did achieve bounds on the error incurred, their complicated formulation makes it difficult to determine whether these bounds are strong. In this section we give strong justification for our construction of the Sensitivity-Preprocessing Function in terms of both error incurred and the exponential running time for the general setting.

In Section 4.4.1 we consider the general problem of approximating an arbitrary function $f : \mathcal{D} \to \mathbb{R}$ with one that has individual sensitivity bounded by $\{\Delta_i\}$. Under the ℓ_∞ metric, our Sensitivity-Preprocessing Function will achieve a 2-approximation of the optimal function. Furthermore, this 2-approximation can still be obtained when the optimal function is restricted to certain subsets of the data universe. Informally, this will imply that on subsets which allow for small error between f and a function with individual sensitivity bounded by $\{\Delta_i\}$, our Sensitivity-Preprocessing Function will also have small error. Due to ℓ_∞ being a worst-case metric, it is then natural to ask if our Sensitivity-Preprocessing Function actually still performs well on the non-worst-case databases. To this end, we show that our Sensitivity-Preprocessing Function is Pareto optimal, meaning that for any

other function with individual sensitivity bounded by $\{\Delta_i\}$, if it has smaller error on some database relative to our Sensitivity-Preprocessing Function, then there must exist another database on which it has higher error.

In Section 4.4.2 we show that it is NP-hard to achieve our approximation guarantees with respect to the ℓ_∞ metric. We further show that it is uncomputable to do better than a 2-approximation in the ℓ_∞ metric, and also uncomputable to achieve even a constant approximation in any ℓ_p metric for $p < \infty$ which justifies our choice of metric. We believe that the combination of these results gives a strong indication that our Sensitivity-Preprocessing Function and corresponding exponential time construction is the best we can hope to achieve for the general problem.

4.4.1 Optimality guarantees

In this section we prove that our Sensitivity-Preprocessing Function achieves certain optimality guarantees. As there are many ways in which to measure how close one function is to another, it is first necessary to be more specific about the definition of optimality we use here. The set that we are trying to optimize over will be all functions with bounded individual sensitivity:

Definition 4.4.1. Given a data universe $\mathcal{D}$ and individual sensitivity parameters $\{\Delta_i\}$, define

$$F_{\{\Delta_i\}}(\mathcal{D}) \stackrel{\text{def}}{=} \{f : \mathcal{D} \to \mathbb{R} \mid \Delta_i(f) \leq \Delta_i, \forall i\}.$$

In this context, the general goal will then be to show that our Sensitivity-Preprocessing Function is close to the optimal function on this set. Here we will consider optimal to be under the ℓ_∞ metric, where we want $f^* \in F_{\{\Delta_i\}}(\mathcal{D})$ to minimize the maximum difference $|f(D) - f^*(D)|$ over all $D \in \mathcal{D}$. Our Sensitivity-Preprocessing Function achieves a 2-approximation to the optimal $f^* \in F_{\{\Delta_i\}}(\mathcal{D})$ with respect to the ℓ_∞ metric. For unbounded sensitivity functions, the value $|f(D) - f^*(D)|$ will be unbounded, so we will instead show

the stronger result that this 2-approximation also holds if we restrict the data universe to a single database and its subsets. Specifically, we show that if for certain subsets of the data universe it is possible to perfectly fit f to a $\{\Delta_i\}$ individual sensitivity bounded function, then our Sensitivity-Preprocessing Function will also perfectly fit to f in this subset. These guarantees are formalized in the following lemma.

Lemma 4.4.2. *Given any $f : \mathcal{D} \to \mathbb{R}$, let $g : \mathcal{D} \to \mathbb{R}$ be the Sensitivity-Preprocessing Function of f with parameters $\{\Delta_i\}$. For any arbitrary $D \in \mathcal{D}$, define $\mathcal{D}' = \{D' \subseteq D\}$. Then,*

$$\max_{D' \in \mathcal{D}'} |f(D') - g(D')| \leq 2 \min_{f^* \in F_{\{\Delta_i\}}(\mathcal{D}')} \max_{D' \in \mathcal{D}'} |f(D') - f^*(D')|$$

Proof. We will prove this inductively on the size of D. It is immediately true for $D = \emptyset$. We now prove for arbitrary D where we assume the claim for all strict subsets of D. Our proof will be by contradiction, where we suppose that our claim is not true for some D.

We first determine the database at which $|f(D') - g(D')|$ is maximized. Suppose $\arg\max_{D' \in \mathcal{D}'} |f(D') - g(D')| = \tilde{D}$ such that $\tilde{D} \subset D$. Define $\tilde{\mathcal{D}} = \{D' \subseteq \tilde{D}\}$. Because $\tilde{D} \subset D$, it must follow that

$$\min_{f^* \in F_{\{\Delta_i\}}(\tilde{\mathcal{D}})} \max_{D' \in \tilde{\mathcal{D}}} |f(D') - f^*(D')| \leq \min_{f^* \in F_{\{\Delta_i\}}(\mathcal{D}')} \max_{D' \in \mathcal{D}'} |f(D') - f^*(D')|.$$

By our assumption that the claim is not true on D, it follows that

$$|f(\tilde{D}) - g(\tilde{D})| > 2 \min_{f^* \in F_{\{\Delta_i\}}(\mathcal{D}')} \max_{D' \in \mathcal{D}'} |f(D') - f^*(D')|.$$

Combining this with the previous inequality implies,

$$|f(\tilde{D}) - g(\tilde{D})| > 2 \min_{f^* \in F_{\{\Delta_i\}}(\tilde{\mathcal{D}})} \max_{D' \in \tilde{\mathcal{D}}} |f(D') - f^*(D')|,$$

which contradicts our inductive hypothesis. Therefore we must have $\max_{D' \in \mathcal{D}'} |f(D') - g(D')| = |f(D) - g(D)|$.

We now apply Lemma 4.4.3, which we prove subsequently, to see that there must exist $\tilde{D} \subset D$ such that $|f(D) - f(\tilde{D})| \geq |f(D) - g(D)| + \sum_{i \in D \setminus \tilde{D}} \Delta_i$. Therefore for any $f^* \in F_{\{\Delta_i\}}(\mathcal{D}')$ it must be true that

$$\max\{|f(\tilde{D}) - f^*(\tilde{D})|, |f(D) - f^*(D)|\} \geq \frac{|f(D) - g(D)|}{2},$$

because of the sensitivity constraints. We then use the fact that $\max_{D' \in \mathcal{D}'} |f(D') - g(D')| = |f(D) - g(D)|$ to conclude,

$$\max_{D' \in \mathcal{D}'} |f(D') - g(D')| \leq 2 \min_{f^* \in F_{\{\Delta_i\}}(\mathcal{D}')} \max_{D' \in \mathcal{D}'} |f(D') - f^*(D')|.$$

This contradicts our assumption, so the claim must therefore be true for D. □

Lemma 4.4.3. *Given any $f : \mathcal{D} \to \mathbb{R}$, let $g : \mathcal{D} \to \mathbb{R}$ be the Sensitivity-Preprocessing Function of f with individual sensitivity parameters $\{\Delta_i\}$. For any $D \in \mathcal{D}$ such that $f(D) \neq g(D)$ there must exist some $\tilde{D} \subset D$ such that $g(D) \geq f(\tilde{D}) + \sum_{i \in D \setminus \tilde{D}} \Delta_i$ if $f(D) > g(D)$ and $g(D) \leq f(\tilde{D}) - \sum_{i \in D \setminus \tilde{D}} \Delta_i$ if $f(D) < g(D)$.*

Proof. We prove the claim inductively, starting with the immediate observation that by construction it is true when D only has one entry.

We now consider an arbitrary D and assume our claim for all subsets. Without loss of generality, we will prove the claim if $f(D) > g(D)$, and can symmetrically apply the proof for the case when $f(D) < g(D)$. If $f(D) > g(D)$, then there must exist some $x_i \in D$ such that $g(D) = g(D - x_i) + \Delta_i$. If $f(D - x_i) \leq g(D - x_i)$, then we can set $\tilde{D} = D - x_i$ and the claim follows. Otherwise we must have $f(D - x_i) > g(D - x_i)$ and we apply our inductive hypothesis to obtain some $\tilde{D} \subset D - x_i$ such that

$$g(D - x_i) \geq f(\tilde{D}) + \sum_{j \in (D - x_i) \setminus \tilde{D}} \Delta_j.$$

We then use the fact that $g(D) = g(D - x_i) + \Delta_i$ to achieve

$$g(D) \geq f(\tilde{D}) + \sum_{j \in D \setminus \tilde{D}} \Delta_j.$$

□

We note that because Lemma 4.4.2 achieves a 2-approximation when the optimal function is restricted to subsets of the data universe, we easily achieve a 2-approximation on the full data universe.

Corollary 4.4.4. *Given any $f : \mathcal{D} \to \mathbb{R}$, let $g : \mathcal{D} \to \mathbb{R}$ be the Sensitivity-Preprocessing Function of f with parameters $\{\Delta_i\}$. Then,*

$$\max_{D' \in \mathcal{D}} |f(D') - g(D')| \leq 2 \min_{f^* \in F_{\{\Delta_i\}}(\mathcal{D})} \max_{D' \in \mathcal{D}} |f(D') - f^*(D')|.$$

Pareto Optimality

We now complement our localized 2-approximation of the ℓ_∞ metric with a Pareto optimality result. As ℓ_∞ is a worst-case metric we would still like our Sensitivity-Preprocessing Function to perform well on the non-worst-case databases. In particular, for the databases that do not contribute to the ℓ_∞ error, we still want the error to be minimized. The following lemma will conclude that we cannot improve the error of a single database without incurring more error on another database, indicating that we are still performing well on the non-worst-case databases.

Lemma 4.4.5. *Given any $f : \mathcal{D} \to \mathbb{R}$, let $g : \mathcal{D} \to \mathbb{R}$ be the Sensitivity-Preprocessing Function of f with individual sensitivity parameters $\{\Delta_i\}$. For any $h \in F_{\{\Delta_i\}}(\mathcal{D})$ if there is some $D \in \mathcal{D}$ such that*

$$|f(D) - h(D)| < |f(D) - g(D)|,$$

then there also exists some $D' \in \mathcal{D}$ such that

$$|f(D') - h(D')| > |f(D') - g(D')|\,.$$

Proof. Suppose there is some $h \in F_{\{\Delta_i\}}(\mathcal{D})$ such that

$$|f(D) - h(D)| < |f(D) - g(D)|$$

for some $D \in \mathcal{D}$, and for all $D' \in \mathcal{D}$,

$$|f(D') - h(D')| \leq |f(D') - g(D')|$$

Then it must be true that $h(\emptyset) = g(\emptyset)$ because $g(\emptyset) = f(\emptyset)$. Let D be the smallest database such that $h(D) \neq g(D)$, which implies that $|f(D) - h(D)| < |f(D) - g(D)|$. This inequality implies $g(D) \neq f(D)$, and by our construction of g, either $\textsc{Upper}(D) < f(D)$ or $\textsc{Lower}(D) > f(D)$.

Without loss of generality, assume $\textsc{Upper}(D) < f(D)$ and thus $g(D) = \textsc{Upper}(D)$. Using the fact that $|f(D) - h(D)| < |f(D) - g(D)|$, we can conclude that $h(D) > g(D)$. However, since $\textsc{Upper}(D) = g(D - x_i) + \Delta_i$ for some $x_i \in D$, we must have $h(D) > g(D - x_i) + \Delta_i$. Our assumption that D was the smallest database such that $h(D) \neq g(D)$ then implies $h(D) > h(D - x_i) + \Delta_i$, contradicting the individual sensitivity of i being at most Δ_i in h.

Therefore, $F_{\{\Delta_i\}}(\mathcal{D})$ cannot contain such an h, which implies our claim.

□

4.4.2 Hardness of approximation

In this section we justify the exponential running time of our implementation of the Sensitivity-Preprocessing Function for the general setting. Recall that in our construc-

tion we did not make any assumptions about $\mathcal{D}$ and only required query access to the function $f : \mathcal{D} \to \mathbb{R}$. Under this limited knowledge setting it is reasonable that our localized greedy construction is the best we can hope for, despite taking exponential time. Accordingly, we show here that even if we restrict $\mathcal{D}$ to be exponential-sized, set all $\{\Delta_i\}$ to be the same Δ, and further force f to be polytime representable, it is still NP-hard to compute our Sensitivity-Preprocessing Function. This proof will further imply that it is NP-hard to compute a function that has identical individual sensitivity guarantees and achieve the same approximation guarantees that our Sensitivity-Preprocessing Function does in Lemma 4.4.2.

After proving this NP-hardness result, we will discuss the issues with computing individual sensitivity bounded functions that obtain better approximations. We give strong justification that it is uncomputable to achieve better than a 2-approximation in the ℓ_∞ metric. Further, we give similar reasons why it is uncomputable to achieve even a constant approximation on average error for the general setting, which justifies our choice of metric for proving our approximation guarantees in the previous section. We believe these ideas could be formalized in a straightforward manner, but think that doing so is unnecessary for the scope of this paper.

NP-hardness

Proposition 4.4.6. *For certain $f : \mathcal{D} \to \mathbb{R}$ such that $|\mathcal{D}| = O(3^n)$, it is NP-hard to compute our Sensitivity-Preprocessing Function g with parameter Δ on a specific database.*

Proof. In order to prove this claim, we will construct a gadget function that takes an arbitrary SAT formula ϕ and constructs a function $f : \mathcal{D} \to \mathbb{R}$ such that $|\mathcal{D}| = O(3^n)$ and on a specified database $D \in \mathcal{D}$, $g(D) < n$ if and only if ϕ is satisfiable. We construct that gadget function below.

Gadget Function: Let $\mathcal{D}$ be the data universe with n individuals such that $x_i \in \{T, F\}$. Let $\phi : \{T, F\}^n \to \{0, 1\}$ be an arbitrary SAT formula of n variables that outputs 0 if false

and 1 if true. For any $D \in \mathcal{D}$, let $D + T \in \{T, F\}^n$ be the assignment of variables that correspond to D and set all variables not in D to be true. Let the function $f_\phi : \mathcal{D} \to \mathbb{R}$ be defined as $f_\phi(D) = |D| - \phi(D + T)$ where $|D| = |\{i \in D\}|$. Further, define $f_\phi(\emptyset) = 0$ and let $\Delta = 1$.

Claim: For the constructed gadget function f from SAT formula ϕ and our corresponding Sensitivity-Preprocessing Function g with parameter Δ, we must have that $g(F^n) < n$ iff ϕ is satisfiable.

First, we assume ϕ is unsatisfiable which implies $f_\phi(D) = |D|$ for all D. Therefore the sensitivity of f is 1, and g will be identical to f, so $g(F^n) = n$.

Next, we show that if $g(F^n) \geq n$ then there cannot exist a satisfying assignment of ϕ. Suppose there does exist a satisfying assignment, then take the one with the fewest false assignments and denote this as $x^* \in \{T, F\}^n$. Further, consider the database $D \subseteq F^n$ that consists of all of the false assignments of x^*. By definition, we must have that $f_\phi(D) = |D| - 1$, and we further show that $g(D) = |D| - 1$.

For any $D' \subset D$, we have $f_\phi(D') = |D'|$ by construction of f_ϕ and our assumption that x^* was the satisfying assignment with the fewest false assignments. It is easy to see that $g(D') = |D'|$ by construction, which implies that $g(D) = |D| - 1$. Since the sensitivity is set to be 1, we have that for every $\tilde{D}$ such that $\tilde{D} \supseteq D$ it must be true that $g(\tilde{D}) \leq |\tilde{D}| - 1$. By construction, we know $D \subseteq F^n$, which implies $g(F^n) < n$. This gives a contradiction and implies that ϕ is unsatisfiable. □

Note that to satisfy the approximation guarantees given in Lemma 4.4.2, any $f^* \in F_\Delta(\mathcal{D})$ would require $f^*(D) = |D| - 1$ in our proof as well. Accordingly, for any $f^* \in F_\Delta(\mathcal{D})$ that satisfies the approximation guarantees of Lemma 4.4.2, it must also be true that $f(F^n) < n$ iff ϕ is satisfiable. Therefore, any algorithm that achieves the same guarantees must also be NP-hard to compute.

Uncomputability of better approximations

We now argue that it is uncomputable to achieve better approximation factors than our Sensitivity-Preprocessing Function, with respect to both the ℓ_∞ metric and any ℓ_p metric.

Remark 4.4.7. We claim that no finitely computable algorithm can obtain a function with appropriately bounded individual sensitivities that achieves better than a 2-approximation on the ℓ_∞ error. Let $\mathcal{D}$ only contain the empty set and databases of size one, each containing a single real-valued data entry $x \in [0, 1]$, and set $\Delta = 1$. Consider any finite algorithm that constructs a Δ-sensitivity function h to minimize the maximum difference between (adversarially chosen) f and h over all databases.

If f is arbitrary and only query accessible, then the algorithm can only query a finite number of databases, and an adversary could just set $f(x) = f(\emptyset) = 0$ for all queried databases. In order to achieve even a constant approximation, the algorithm would need to set $f(x) = 0$ just in case $f(x) = 0$ for all $x \in [0, 1]$. However, the adversary could then set $f(y) = 2$ for all non-queried databases. The function that minimizes the ℓ_∞ error would then set $f(x) = 1/2$ for all queried databases and $f(y) = 3/2$ for all non-queried databases. As a result, the finite algorithm can only achieve a 2-approximation.

Remark 4.4.8. We further claim that no finitely computable algorithm can obtain a function with appropriately bounded individual sensitivities that achieves a constant approximation on the average ℓ_p error. The optimal function in this scenario would be f^* that minimizes:

$$\min_{f^* \in F_{\{\Delta_i\}}(\mathcal{D})} \left(\frac{\sum_{D \in \mathcal{D}} (f(D') - f^*(D'))^p}{|\mathcal{D}|} \right)^{1/p}.$$

We consider the same example as above, and note that the number of queried databases is finite and the number of non-queried databases is infinite. In order to achieve a constant approximation, the algorithm would need to set $f(x) = 0$ just in case $f(x) = 0$ for all $x \in [0, 1]$. However, it would then have to set $f(y) = 1$ for all non-queried databases and the average ℓ_p error would be a constant. If instead it set $f(x) = 1$ for all queried databases

and $f(y) = 2$ for all non-queried databases, then the average ℓ_p error would approach 0 because the non-queried databases are infinite and the queried databases are finite. As a result, no finitely computable algorithm can achieve a constant approximation in this metric.

4.5 Efficient Implementation of Several Statistical Measures

In this section, we take our general recursive algorithm and show how it can be made efficient for a variety of important statistical measures such as mean, α-trimmed mean, median, minimum, and maximum. It is important to note that we will not change the key recursive structure, but instead show that when we have more information about the function, we can ignore many of the subproblems of the recursion for significant runtime speedups. As a result, the algorithm given for these statistical tasks will take $O(n^2)$ time and have a simple dynamic programming construction.

The key idea will be that given a database $D = (x_1, ..., x_n)$ where we assume for simplicity that $x_1 \leq \cdots \leq x_n$,[5] the only important subproblems will be $D - x_1$ and $D - x_n$. Consequently, instead of considering every possible subset of D, we only need to consider every contiguous subset, which limits the number of subproblems to $O(n^2)$.

We first give a general class of functions—which includes mean, median, α-trimmed mean, minimum, and maximum—for which it is straightforward to show our algorithm can be applied efficiently. We then give a more in-depth analysis of the error guarantees that correspond with this implementation for mean. These bounds will ultimately be quite intuitive, but the proofs will be more involved.

4.5.1 Efficient implementation for a simple class of functions

We will first define a class of functions under which database ordering is preserved for any subset, which allows us to presort the data according to this ordering and restrict the number of subproblems. Intuitively, it implies that for any database $D = (x_1, ..., x_n)$ there is an

[5] Our algorithm will presort and only incur $O(n \log n)$ running time.

ordering of the $x_1, ..., x_n$ such that the extreme points in our recursion are determined by the databases that remove the maximum or the minimum. In particular, if we consider the mean function $\mu : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ then for any $D = (x_1, ..., x_n)$ if we assume $x_1 \leq \cdots \leq x_n$, then we know $\mu(D - x_n) \leq \mu(D - x_i)$ and $\mu(D - x_1) \geq \mu(D - x_i)$ for any i. This will ultimately imply that our upper and lower bounds on the allowable region for $g(D)$ will be defined by $g(D - x_1)$ and $g(D - x_n)$, respectively.

Definition 4.5.1 (Database-ordered function)**.** A function $f : \mathcal{D} \to \mathbb{R}$ is *database-ordered* if for any $D = (x_1, ..., x_n) \in \mathcal{D}$ and any pair $x_i, x_j \in D$, we have that for every subset database $D' \subset D$ such that $x_i, x_j \notin D'$, then either $f(D' + x_i) \leq f(D' + x_j)$ for every D' or $f(D' + x_i) \geq f(D' + x_j)$ for every D'. Furthermore, if $f(D' + x_i) \leq f(D' + x_j)$ for every D', we say that $x_i \leq x_j$ in the *entry-ordering*, and vice-versa if $f(D' + x_i) \geq f(D' + x_j)$ for every D'.

The general idea of our efficient implementation will be to use the ordering and only consider contiguous subsets according to this ordering.

Lemma 4.5.2. *Given a database-ordered function $f : \mathcal{D} \to \mathbb{R}$, let $g : \mathcal{D} \to \mathbb{R}$ be the Sensitivity-Preprocessing Function of f with parameter Δ. Then for any $D = (x_1, ..., x_n)$ where $x_1 \leq \cdots \leq x_n$ in the entry-ordering we must have* $\textsc{Upper}(D) = g(D - x_n) + \Delta$ *and* $\textsc{Lower}(D) = g(D - x_1) - \Delta$*, and our* PREPROCESSING *algorithm only requires solving $O(n^2)$ subproblems*

Proof. We first want to show $\textsc{Upper}(D) = g(D - x_n) + \Delta$ and $\textsc{Lower}(D) = g(D - x_1) - \Delta$. It is sufficient to show $g(D - x_n) \leq g(D - x_{n-1}) \leq \cdots \leq g(D - x_1)$, which we will prove by induction on the size of the database. If D only has one entry, then this must be true.

Assume this is true for all D with at most $n - 1$ entries, and we want to show $g(D - x_{i+1}) \leq g(D - x_i)$ for any $i \in [n - 1]$. Since f is database-ordered, we know that $f(D - x_{i+1}) \leq f(D - x_i)$. It then suffices to show $\textsc{Upper}(D - x_{i+1}) \leq \textsc{Upper}(D - x_i)$ and $\textsc{Lower}(D - x_{i+1}) \leq \textsc{Lower}(D - x_i)$. By our inductive hypothesis, $\textsc{Upper}(D - x_i) =$

$g(D - x_i - x_n) + \Delta$ and $\textsc{Upper}(D - x_{i+1}) = g(D - x_{i+1} - x_n) + \Delta$ if $i < n - 1$, and we note that $\textsc{Upper}(D - x_{n-1}) = \textsc{Upper}(D - x_n)$. Also by our inductive hypothesis, $g(D - x_{i+1} - x_n) \leq g(D - x_i - x_n)$, implying $\textsc{Upper}(D - x_{i+1}) \leq \textsc{Upper}(D - x_i)$. The proof for $\textsc{Lower}(D - x_{i+1}) \leq \textsc{Lower}(D - x_i)$ follows symmetrically.

With this fact, it is straightforward to see that for our algorithm, instead of considering all subsets of size k, it suffices to consider $(x_1, \ldots, x_k), (x_2, \ldots, x_{k+1}), \ldots, (x_{n-k}, \ldots, x_n)$. Then the total number of subproblems that need to be solved is $O(n^2)$. □

If our function is efficiently computable and the entry-ordering is efficiently computable, this then gives an efficient implementation of our recursive algorithm. In particular, for several functions of statistical interest including mean, α-trimmed mean, median, maximum, and minimum, this easily yields an efficient algorithm.

Algorithm 11: Efficient Implementation for database-ordered functions

Input: Database-ordered function $f : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$, sensitivity bound Δ, estimate for the empty set $\hat{\mu}$, and database $D = (x_1, ..., x_n) \in \mathbb{R}^n$ for some arbitrary n.
Output: $g(D)$, where g is the Sensitivity-Preprocessing Function of f.
Initialize $g(\emptyset) = \hat{\mu}$
Sort D (We will assume $x_1 \leq \cdots \leq x_n$ for simplicity)
For $k = 1, \ldots, n$
For $i = 1, \ldots, n - k + 1$
For every database $D' = (x_i, ..., x_{i+k-1})$

$$\text{Let } g(D') = \begin{cases} g(D' - x_{i+k-1}) + \Delta, & \text{if } g(D' - x_{i+k-1}) + \Delta \leq f(D') \\ g(D' - x_i) - \Delta, & \text{if } g(D' - x_i) - \Delta \geq f(D') \\ f(D'), & \text{otherwise} \end{cases}$$

Output $g(D)$

Corollary 4.5.3. *We can implement our Sensitivity-Preprocessing Function with parameter Δ in $O(n^2)$ time for the functions mean, α-trimmed mean, median, maximum, and minimum.*

Proof. Let f be any of the functions listed above. It is simple to see that for any $D = (x_1, ..., x_n) \in \mathbb{R}^n$, and any $y, z \in \mathbb{R}$, if $y \leq z$ then $f(D + y) \leq f(D + z)$, and if $y \geq z$ then $f(D + y) \geq f(D + z)$. This implies that f is database-ordered, then by Lemma 4.5.2 we only need to solve $O(n^2)$ subproblems.

Further, we note that finding the entry-ordering simply requires sorting the entries of D in $O(n \log n)$ time. If the database is ordered, then computing median, minimum, and maximum only requires $O(1)$ time. If we know the mean or α-trimmed mean for $D - x_i$ for some x_i, we can compute the mean or α-trimmed mean of D in $O(1)$ time using the fact that

$$\frac{x_1 + ... + x_n}{n} = \frac{n-1}{n}\left(\frac{x_1 + ... + x_{n-1}}{n-1}\right) + \frac{x_n}{n}$$

Note that we compute $D - x_i$ for some i in our subproblems, so we will in fact have access to this value. As a result, the full running time will take $O(n^2)$ time.

□

4.5.2 Improved runtime and accuracy for median

In the previous section, we showed that for several important statistical measures we could give a simple efficient version of our general algorithm. To complement this result, we further examine the median function and give an improved analysis that requires only $O(n)$ time for presorted data and provides strong accuracy guarantees. Improving the running time will utilize the critical property that removing the minimum and maximum value does not change the median. As was seen in our previous section, our recursion was reduced by only considering removing the maximum or minimum value. The related fact regarding median will be incorporated into an inductive claim that we never overshoot the true median, and can further reduce our recursion.

Lemma 4.5.4. *Let $med : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ be the median function and $g : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ be the Sensitivity-Preprocessing Function of med with parameter Δ. Then for any $D = (x_1, ..., x_n)$ such that $x_1 \leq \cdots \leq x_n$, computing $g(D)$ takes $O(n)$ time.*

Proof. It follows immediately from Lemma 4.5.2 and Lemma 4.5.5 that if $med(D) \geq med(\emptyset)$ then $g(D) = \min\{med(D), g(D - x_n) + \Delta\}$ and otherwise we must have $g(D) = \max\{med(D), g(D - x_1) - \Delta\}$. We can calculate $med(D)$ and any contiguous subset of

D in $O(1)$ time, and the recursion will only be upon one subproblem, implying a runtime of $O(n)$. □

Lemma 4.5.5. *If* $med(D) \geq med(\emptyset)$*, then* $med(\emptyset) \leq g(D) \leq med(D)$

Proof. The proof will be inductive, and it is easy to verify that the inequality holds for $|D| \leq 2$. We then consider an arbitrary $D = (x_1, ..., x_n)$ where we assume without loss of generality that $x_1 \leq \cdots \leq x_n$ and $n \geq 3$. The critical fact we use here will be that the median does not change if you remove the minimum and maximum values, which is to say that $med(D) = med(D - x_1 - x_n)$. Therefore, if $med(D) \geq med(\emptyset)$, then we must also have $med(D - x_1 - x_n) \geq med(\emptyset)$, which by our inductive claim implies that $med(\emptyset) \leq g(D - x_1 - x_n) \leq med(D - x_1 - x_n) = med(D)$. Applying Lemma 4.5.2, we then have

$$g(D - x_1) \leq g(D - x_1 - x_n) + \Delta \leq med(D) + \Delta$$

and

$$g(D - x_n) \geq g(D - x_1 - x_n) - \Delta \geq med(D) - \Delta$$

We then reapply Lemma 4.5.2 to achieve our desired result that $med(\emptyset) \leq g(D) \leq med(D)$ □

As in [11], define $A^{(k)}(D) = \max_{0 \leq t \leq k+1}(x_{m+t} - x_{m+t-k-1})$ and $m = \frac{n+1}{2}$, which is essentially the k-local sensitivity of median for database D. More formally,

$$A^{(k)}(D) = \max_{d(D,D') \leq k} LS_f(D').$$

Combining this assumption with our previous lemma will then allow for stronger bounds upon $g(D)$.

Lemma 4.5.6. *Given some parameter* Δ *and* $med(\emptyset)$*, if* $A^{(k)}(D) \leq 2(k+1)\Delta$ *for* $k \leq n/4$ *and* $med(D) \in [med(\emptyset) - \frac{n}{2}\Delta, med(\emptyset) + \frac{n}{2}\Delta]$*, then* $g(D) = med(D)$

Proof. Without loss of generality, assume that $med(D) \geq med(\emptyset)$. By Lemma 4.5.5 we know $g(D) \leq med(D)$, then applying Lemma 4.5.7 gives our desired result. □

Lemma 4.5.7. *Given some parameter Δ and $med(\emptyset)$, assume $A^{(k)}(D) \leq 2(k+1)\Delta$ for $k \leq n/4$ and $med(D) \in [med(\emptyset) - \frac{n}{2}\Delta, med(\emptyset) + \frac{n}{2}\Delta]$. Let $D_{[1:k]} = (x_1, ..., x_k)$, if $med(D) \geq med(\emptyset)$, then $g(D_{[1:k]}) \geq med(D) - (n-k)\Delta$*

Proof. It is immediately implied by our assumptions that

$$med(D_{[1:k]}) \geq med(D) - (n-k)\Delta$$

for any $k \geq n/2$. We then consider our base case to be $k = n/2$, and note that from Lemma 4.5.5 we have $g(D_{[1:k]}) \geq \min\{med(\emptyset), med(D_{[1:k]})$, which by our assumptions immediately implies $g(D_{[1:n/2]}) \geq med(D) - \frac{n}{2}\Delta$.

We then assume this is true for $k-1 \geq n/2$, so $g(D_{[1:k-1]}) \geq med(D) - (n-k)\Delta - \Delta$. We then also know from Lemma 4.5.2 that

$$g(D_{[1:k]}) \geq \min\{med(D_{[1:k]}), g(D_{[1:k-1]}) + \Delta\}$$

which implies our desired inequality.

□

Proof of Theorem 4.1.9

We now have all the necessary components to give our proof of Theorem 4.1.9, which we restate and prove below.

Theorem 4.1.9. *Let $med : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ be the median function for the data universe of all finite-length real-valued vectors. For chosen parameters $med(\emptyset)$ and Δ, along with any database $D = (x_1, ..., x_n) \in \mathbb{R}^{<\mathbb{N}}$, if $x_1 \leq \cdots \leq x_n$ we give $O(n)$ time access to a function $g : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ with sensitivity Δ such that $g(D) = med(D)$ whenever $A^{(k)}(D) \leq 2(k+1)\Delta$ for $k \leq n/4$ and $med(D) \in [med(\emptyset) - \frac{n}{2}\Delta, med(\emptyset) + \frac{n}{2}\Delta]$.*

Proof of Theorem 4.1.9. The runtime guarantees follow immediately from Lemma 4.5.4. Furthermore, if we assume that $med(D) \geq med(\emptyset)$, then Lemma 4.5.5 implies that $g(D) \leq med(D)$ and Lemma 4.5.7 implies that $g(D) \geq med(D)$ because we have the same assumptions, and $g(D) = med(D)$ The symmetric version of these lemmas follows immediately, and we also have $g(D) = med(D)$ when $med(D) \leq med(\emptyset)$.

□

4.5.3 Accuracy bounds for mean

We next consider the mean function, and provide strong bounds on the accuracy of our Sensitivity-Preprocessing Function. While the analysis will be rather involved, we believe that the ultimate guarantees are highly intuitive. Our proof will also show that for databases with entries bounded in a Δ sensitivity range, we perfectly preserve the accuracy between our new function and the mean function. Further, the key ideas in our proof are closely related to the construction of our recursive function, and we believe could be extended to other functions using a similar framework.

The general proof idea will be to give two simpler recursive functions that yield reasonably tight upper and lower bounds on our function. Due to their further simplicity, it will be much easier to give nice error bounds with respect to the true mean for these functions.

The idea behind constructing the upper and lower bound functions will be simple. Recall that we showed our g for the mean function has the property that $\textsc{Upper}(D) = g(D-x_n)+\Delta$ and $\textsc{Lower}(D) = g(D-x_1)-\Delta$ because we showed $g(D-x_n) \leq g(D-x_{n-1}) \leq \cdots \leq g(D-x_1)$ if we assume $x_1 \leq \cdots \leq x_n$. Intuitively, this is due to the fact that removing the maximum value will minimize mean and removing the minimum value will maximize mean. Accordingly, we will just iteratively remove the maximum value to give a lower bound on our function and iteratively remove the minimum value to give an upper bound on our function. These functions then only require solving $O(n)$ subproblems which will simplify the analysis.

Definition 4.5.8 (Mean-bounding functions)**.** For any $D = (x_1, ..., x_n)$, define

$$h_{lower}(D) = \begin{cases} h_{lower}(D - x_n) + \Delta, & \text{if } h_{lower}(D - x_n) + \Delta \leq \mu(D) \\ h_{lower}(D - x_n) - \Delta, & \text{if } h_{lower}(D - x_n) - \Delta \geq \mu(D) \\ \mu(D), & \text{otherwise} \end{cases}$$

and

$$h_{upper}(D) = \begin{cases} h_{upper}(D - x_1) + \Delta, & \text{if } h_{upper}(D - x_1) + \Delta \leq \mu(D) \\ h_{upper}(D - x_1) - \Delta, & \text{if } h_{upper}(D - x_1) - \Delta \geq \mu(D) \\ \mu(D), & \text{otherwise} \end{cases}$$

We will first show that h_{upper} and h_{lower} are upper and lower bounds, respectively, of our Sensitivity-Preprocessing Function g with parameter Δ. Then we further examine the properties of these functions.

Lemma 4.5.9. *Let $\mu : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ be the mean function with chosen parameters $\hat{\mu}$ and Δ. For any $D = (x_1, ..., x_n) \in \mathbb{R}^n$ with $x_1 \leq x_2 \leq \cdots \leq x_n$, then $h_{lower}(D) \leq g(D)$ and $h_{upper}(D) \geq g(D)$ where $g : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ is our Sensitivity-Preprocessing Function with parameter Δ.*

Proof. We will prove both inequalities by induction, where we first note that if D only has one entry, then by construction $h_{lower}(D) = g(D) = h_{upper}(D)$.

For any database D of n entries, by induction we have $h_{lower}(D - x_n) \leq g(D - x_n)$ and note that within the proof of Lemma 4.5.2 we showed $g(D - x_n) \leq g(D - x_1)$, which implies $h_{lower}(D) \leq g(D)$. Similarly, by induction we have $h_{upper}(D - x_1) \geq g(D - x_1)$ and Lemma 4.5.2 gives $g(D - x_1) \geq g(D - x_n)$, which implies $h_{upper}(D) \leq g(D)$. □

We now use the simpler recursive structure of h_{lower} and h_{upper} to get more explicit forms of their output.

Lemma 4.5.10. *Let $\mu : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ be the mean function with chosen parameters $\hat{\mu}$ and Δ. For any $D = (x_1, ..., x_n)$, assume that $x_1 \leq x_2 \leq \cdots \leq x_n$, and let $D_{[i:j]} = (x_i, ..., x_j)$. Let k be the largest index such that $h_{lower}(D_{[1:k]}) \geq \mu(D_{[1:k]})$ (if one exists), then*

$$h_{lower}(D_{[1:k]}) = \max\{\hat{\mu} - k\Delta, \mu(D[1:k])\}.$$

Let l be the smallest index such that $h_{upper}(D_{[l:n]}) \geq \mu(D_{[l:n]})$ (if one exists), then

$$h_{upper}(D_{[l:n]}) = \min\{\hat{\mu} + (n-l)\Delta, \mu(D_{[l:n]})\}.$$

Proof. We consider the first equality here, and the second follows symmetrically.

Note that $\mu(D_{[1:k]})$ is increasing in k because $x_1 \leq \cdots \leq x_n$. By construction of h_{lower}, if for some index k' we have $h_{lower}(D_{[1:k']}) \leq \mu(D_{[1:k']})$, then $h_{lower}(D_{[1:k'+1]}) \leq \mu(D_{[1:k'+1]})$. Accordingly, if we let k_{min} be the first index such that $h_{lower}(D_{[1,k_{min}]}) \leq \mu(D_{1,k_{min}})$, then in the case that $k \geq k_{min}$ we must have $h_{lower}(D_{[1:k]}) = \mu(D_{[1:k]})$. If $k < k_{min}$, then we must have $h_{lower}(D_{[1:k]}) > \mu(D_{[1:k]})$, and furthermore $h_{lower}(D_{[1:k']}) > \mu(D_{[1:k']})$ for all $k' \leq k$, which implies that we always decreased by Δ and we get $h_{lower}(D_{[1:k]}) = \hat{\mu} - k\Delta$. □

We use the explicit forms of h_{lower} and h_{upper} to sandwich the loss in accuracy, by considering the inflection point of $n/3$ and bounding the error from h_{lower} separately for $k \leq n/3$ and for $k \geq n/3$. The analogous result follows symmetrically for h_{upper}.

Lemma 4.5.11. *Let $\mu : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ be the mean function with chosen parameters $\hat{\mu}$ and Δ. If $g : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ is our Sensitivity-Preprocessing Function with parameter Δ, then given any $D = (x_1, ..., x_n)$,*

$$|g(D) - \mu(D)| \leq \max\{|\hat{\mu} - \mu(D)| - \frac{n}{3}\Delta, 0\} + \sum_{i=1}^{n} \max\left\{\frac{27\,|x_i - \mu(D)|}{n} - \Delta, 0\right\}.$$

Proof. If we can instead prove the same upper bounds for both $|h_{lower}(D) - \mu(D)|$ and $|h_{upper}(D) - \mu(D)|$, then the desired bound for $|g(D) - \mu(D)|$ follows from Lemma 4.5.9.

We give the desired bound for $|h_{lower}(D) - \mu(D)|$, and the bound for $|h_{upper}(D) - \mu(D)|$ follows symmetrically.

Again, let k be the largest index such that $h_{lower}(D_{[1:k]}) \geq \mu(D_{[1:k]})$ (if one exists). If $k \leq n/3$ or none exists, then it immediately follows from Lemma 4.5.10 that $h_{lower}(D) \geq \hat{\mu} + \frac{n}{3}\Delta$, which implies $|\mu(D) - h_{lower}(D)| \leq |\mu(D) - \hat{\mu}| - \frac{n}{3}\Delta$.

If $k \geq n/3$, then it is implied by Lemma 4.5.10 that $h_{lower}(D) = \max\{\hat{\mu} - k\Delta, \mu(D_{[1:k]})\} + (n-k)\Delta \geq \mu(D_{[1:k]}) + (n-k)\Delta$ and therefore,

$$\mu_D - h_{lower}(D) \leq \mu(D) - \mu(D_{[1:k]}) + (n-k)\Delta = \sum_{i=k}^{n-1} \left(\mu(D_{[1:i+1]}) - \mu(D_{[1:i]})\right) - (n-k)\Delta.$$

Furthermore,

$$\mu(D_{[1:i+1]}) - \mu(D_{[1:i]}) = \frac{x_1 + \cdots + x_{i+1}}{i+1} - \frac{x_1 + \cdots + x_i}{i} = \frac{1}{i(i+1)} \left(\sum_{j=1}^{i} x_{i+1} - x_j \right).$$

We use the fact that $i \geq n/3$ to achieve,

$$\mu(D) - h_{lower}(D) \leq \left(\frac{9}{n^2} \sum_{i=k}^{n} \sum_{j=1}^{i} (x_i - x_j) \right) - (n-k)\Delta.$$

Applying Lemma 4.5.12 (stated below) gives,

$$\mu(D) - h_{lower}(D) \leq \left(\frac{27}{n} \sum_{i=k}^{n} |x_i - \mu(D)| \right) - (n-k)\Delta = \sum_{i=k}^{n} \left(\frac{27\,|x_i - \mu(D)|}{n} - \Delta \right)$$

We then add in non-negative terms that are necessary for the symmetric version with h_{upper} to achieve our desired bound. □

We used the following lemma to simplify the bounds in Lemma 4.5.11 beyond those stated in the more general Lemma 4.3.6. We relegate the proof of this lemma to the appendix.

Lemma 4.5.12. *For any set of reals $D = (x_1, ..., x_n)$ where $x_1 \leq \cdots \leq x_n$, given any index*

$k \in [n]$,

$$\frac{1}{n^2}\sum_{i=k}^{n}\sum_{j=1}^{i}\frac{1}{3}\left|x_i - x_j\right| \leq \frac{1}{n}\sum_{i=k}^{n}\left|x_i - \mu(D)\right|.$$

To finally obtain all the necessary components for the proof of Theorem 4.1.10, it is only left to show that when all the inputs of the database are in a nicely bounded range, our Sensitivity-Preprocessing Function will perfectly fit to the function μ.

Lemma 4.5.13. *Let $\mu : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ be the mean function with chosen parameters $\hat{\mu}$ and Δ. If $g : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ is our Sensitivity-Preprocessing Function with parameter Δ, then given any $D = (x_1, ..., x_n) \in \mathbb{R}^n$, if for all $x_i \in D$ we have $x_i \in [\hat{\mu} + \alpha\Delta, \hat{\mu} + (\alpha + n)\Delta]$ for $\alpha \in [-n, 0]$, then $g(D) = \mu(D)$.*

Proof. First, it is straightforward to see by the construction of h_{lower} and h_{upper} that $h_{lower}(D) \leq \mu(D)$ if $\mu(D) \geq \hat{\mu} - n\Delta$ and $h_{upper}(D) \geq \mu(D)$ if $\mu(D) \leq \hat{\mu} + n\Delta$. Therefore, by Lemma 4.5.9, the desired result is implied if $h_{lower}(D) \geq \mu(D)$ and $h_{upper}(D) \leq \mu(D)$. Here we show that $h_{lower}(D) \geq \mu(D)$, and $h_{upper}(D) \leq \mu(D)$ will be implied symmetrically.

Suppose it is not true that $h_{lower}(D) \geq \mu(D)$, then there must exist the last index $k < n$ such that $h_{lower}(D_{[1:k]}) \geq \mu(D_{[1:k]})$, which by construction implies that $h_{lower}(D) = h_{lower}(D_{[1:k]}) + (n - k)\Delta$. To achieve our contradiction, we want to show that $\mu(D) - h_{lower}(D_{[1:k]}) \leq (n - k)\Delta$.

By our restriction of each x_i and by assumption we have,

$$\hat{\mu} + \alpha\Delta \leq \mu(D_{[1:k]}) \leq h_{lower}(D_{[1:k]}).$$

Furthermore, because all of the remaining $x_i \leq \hat{\mu} + (\alpha + n)\Delta$, we must have,

$$\mu(D) \leq \frac{k\mu(D_{[1:k]}) + (n - k)(\hat{\mu} + (\alpha + n)\Delta)}{n} \leq \frac{k \cdot h_{lower}(D_{[1:k]}) + (n - k)(\hat{\mu} + (\alpha + n)\Delta)}{n},$$

where the second inequality follows from our assumption that $h_{lower}(D_{[1:k]}) \geq \mu(D_{[1:k]})$.

This implies,

$$\begin{aligned}\mu(D) - h_{lower}(D_{[1:k]}) &\leq \frac{k \cdot h_{lower}(D_{[1:k]}) + (n-k)(\hat{\mu} + (\alpha+n)\Delta)}{n} - h_{lower}(D_{[1:k]}) \\ &= \frac{(k-n)h_{lower}(D_{[1:k]}) + (n-k)(\hat{\mu} + \frac{n}{2}\Delta)}{n}\end{aligned}$$

We use the fact that $h_{lower}(D_{[1:k]}) \geq \hat{\mu} + \alpha\Delta$ and $k < n$ to get,

$$\mu(D) - h_{lower}(D_{[1:k]}) \leq \frac{(k-n)(\hat{\mu} + \alpha\Delta) + (n-k)(\hat{\mu} + (\alpha+n)\Delta)}{n} = (n-k)\Delta,$$

giving our desired contradiction, which implies $h_{lower}(D) \geq \mu(D)$. □

Proof of Theorem 4.1.10

We now have all the necessary components to give our proof of Theorem 4.1.10, which we restate and prove below.

Theorem 4.1.10. *Let $\mu : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ be the mean function for the data universe of all finite-length real-valued vectors. For chosen parameters $\hat{\mu}$ and Δ, along with any database $D = (x_1, ..., x_n) \in \mathbb{R}^{<\mathbb{N}}$, we give $O(n^2)$ time access to a function $g : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ with sensitivity Δ such that,*

$$|g(D) - \mu(D)| \leq \max\left\{|\mu(D) - \hat{\mu}| - \frac{n}{3}\Delta, 0\right\} + \sum_{i=1}^{n} \max\left\{\frac{27\,|x_i - \mu(D)|}{n} - \Delta, 0\right\}.$$

Additionally, if we are guaranteed that each $x_i \in [\hat{\mu}+\alpha\Delta, \hat{\mu}+(\alpha+n)\Delta]$ for $\alpha \in [-n, 0]$, then $g(D) = \mu(D)$

Proof of Theorem 4.1.10. The fact that g has sensitivity Δ follows from the fact that it is our Sensitivity-Preprocessing Function and the guarantees of Lemma 4.3.5. The runtime follows from Corollary 4.5.3. We then achieve the error bounds from Lemma 4.5.11 and Lemma 4.5.13.

□

4.6 Efficient Implementation for Variance

In this section, we show how to efficiently extend our recursive algorithm to *variance*, which is an important statistical metric and a more complicated function than those considered in Section 4.5. The general idea will remain the same as we reduce the number of subproblems to $O(n^2)$ by using structural properties of variance. We first formally define the discrete version of variance with two equivalent equations.

Definition 4.6.1. For any $D = (x_1, ..., x_n) \in \mathbb{R}^n$, let $\mu(D) = \frac{1}{n}(x_1 + \cdots + x_n)$ and define the variance function,

$$\textbf{Var}\left[D\right] \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \mu(D)\right)^2,$$

or equivalently,

$$\textbf{Var}\left[D\right] \stackrel{\text{def}}{=} \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{1}{2}\left(x_i - x_j\right)^2.$$

As with mean, α-trimmed mean, median, maximum, and minimum, we will first sort the entries of the database. Intuitively, we can decrease the variance most by removing either the minimum or maximum value. We make use of the following fact, which we prove in the appendix for completeness.

Fact 4.6.2. *Given $D = (x_1, ..., x_n) \in \mathbb{R}^n$ such that $x_1 \leq \cdots \leq x_n$, then for any i,*

$$\min\{\textbf{Var}\left[D - x_1\right], \textbf{Var}\left[D - x_n\right]\} \leq \textbf{Var}\left[D - x_i\right].$$

We will use this fact to show that the lower bound on $g(D)$ will be defined by $g(D - x_1)$ or $g(D - x_n)$. The difficulty now becomes that to increase variance the most, we would want to remove an entry between x_1 and x_n. This poses a significant complication in constructing a dynamic program for the subproblems. More specifically, even if $g(D)$ only required solving two subproblems $g(D - x_i)$ and $g(D - x_j)$ for some x_i, x_j, we are still doubling the number of subproblems at each step. The straightforward dynamic program for

ordered-databases was able to reuse different subproblems to avoid a runtime blow-up. The key idea will then be that we can bound, with respect to the original variance, the amount variance can be increase by removing an entry. In particular, we use the following fact that is likely a folklore result, but we could not find a citation, so we prove it in the appendix for completeness.

Fact 4.6.3. *Given any unordered* $(x_1, ..., x_n) \in \mathbb{R}^n$*,*

$$\textbf{Var}\left[x_1, ..., x_{n-1}\right] \leq \frac{n}{n-1}\textbf{Var}\left[x_1, ..., x_n\right].$$

We can then use this strong bound to show that if we initialize $g(\emptyset) = 0$, the Sensitivity-Preprocessing Function will never go above $\textbf{Var}\left[D\right]$ for any $g(D)$. As a result, the Sensitivity-Preprocessing Function will never actually use $\textsc{Lower}(D)$. This will then allow us to only recurse on subproblems where the minimum or maximum has been removed, and the dynamic program will be analogous to the one given for mean.

We first give the efficient implementation for variance and show that it can be done in $O(n^2)$ time. Then we give stronger bounds on the error incurred by this efficient implementation, and finally use these facts to prove Theorem 4.1.11.

4.6.1 Efficient algorithm for variance

As with mean and the database-ordered functions, the key to our efficient implementation will be showing that the Sensitivity-Preprocessing Function can be equivalently defined using far fewer subproblems. Using some of the intuition above, we are able to prove the following lemma that reduces the Sensitivity-Preprocessing Function to a much simpler recursion.

Lemma 4.6.4. *Let* $\textbf{\textit{Var}} : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ *be the variance function and set* $\textbf{Var}\left[\emptyset\right] = 0$*. Then the Sensitivity-Preprocessing Function with parameter* Δ *can be equivalently defined as* $g(\emptyset) = 0$ *and* $g(D) = \min\{\textbf{Var}\left[D\right], g(D - x_1) + \Delta, g(D - x_n) + \Delta\}$ *where* $D = (x_1, ..., x_n)$

with $x_1 \leq \cdots \leq x_n$.

We will prove this lemma with the following two helper lemmas. The first will show that the Sensitivity-Preprocessing Function will never exceed the true variance. The second uses the fact that variance is minimized by either removing the minimum or maximum value to show that the lower bound can simply consider the subproblems $g(D - x_1)$ and $g(D - x_n)$.

Lemma 4.6.5. *Given any* $D = (x_1, ..., x_n) \in \mathbb{R}^n$*, if* g *is the Sensitivity-Preprocessing Function of variance with parameter* Δ *and* $g(\emptyset) = 0$*, then,*

$$g(D) \leq \textbf{Var}\left[D\right].$$

Proof. We will prove this by induction. If D contains only a single entry, then $\textbf{Var}\left[D\right] = 0$ and by construction $g(D) = 0$.

We then consider $D = (x_1, ..., x_n)$ and assume the inequality holds for all subsets. By the definition of the Sensitivity-Preprocessing Function, it suffices to show that $g(D - x_i) - \Delta \leq \textbf{Var}\left[D\right]$ for all x_i. Our inductive claim gives $g(D - x_i) \leq \textbf{Var}\left[D - x_i\right]$, and Fact 4.6.3 implies:

$$\textbf{Var}\left[D - x_i\right] - \textbf{Var}\left[D\right] \leq \frac{1}{n-1}\textbf{Var}\left[D\right],$$

These combine to give,

$$g(D - x_i) - \textbf{Var}\left[D\right] \leq \frac{1}{n-1}\textbf{Var}\left[D\right].$$

We now consider two cases. If $g(D - x_i) \leq \textbf{Var}\left[D\right]$, then $g(D - x_i) - \Delta \leq \textbf{Var}\left[D\right]$ because $\Delta \geq 0$ and we have our desired inequality. If $\textbf{Var}\left[D\right] \leq g(D - x_i)$ then,

$$g(D - x_i) - \textbf{Var}\left[D\right] \leq \frac{1}{n-1}g(D - x_i).$$

Further, by the definition of Sensitivity-Preprocessing Function and the fact that $g(\emptyset) = 0$,

we must have $g(D - x_i) \leq (n-1)\Delta$, implying,

$$g(D - x_i) - \textbf{Var}\left[D\right] \leq \Delta,$$

which is our desired inequality. □

Lemma 4.6.6. *Given $D = (x_1, ..., x_n) \in \mathbb{R}^n$ such that $x_1 \leq \cdots \leq x_n$ and g is the Sensitivity-Preprocessing Function of variance with parameter Δ and $g(\emptyset) = 0$, then,*

$$\min\{g(D - x_1), g(D - x_n)\} \leq g(D - x_i),$$

for any $x_i \in D$.

Proof. We will prove this by induction. If D has just one entry then $x_1 = x_i = x_n$ and each term is equivalent.

We then consider $D = (x_1, ..., x_n)$ and assume the inequality holds for all subsets. We will consider two cases. Our first case is $g(D - x_i) = \textbf{Var}\left[D - x_i\right]$. Lemma 4.6.5 implies:

$$\min\{g(D - x_1), g(D - x_n)\} \leq \min\{\textbf{Var}\left[D - x_1\right], \textbf{Var}\left[D - x_n\right]\}.$$

Furthermore, by Fact 4.6.2 we have $\min\{\textbf{Var}\left[D - x_1\right], \textbf{Var}\left[D - x_n\right]\} \leq \textbf{Var}\left[D - x_i\right]$. Combining this with the assumption $g(D - x_i) = \textbf{Var}\left[D - x_i\right]$ gives the desired inequality.

It is implied by Lemma 4.6.5 that the only other case we need to consider is $g(D - x_i) < \textbf{Var}\left[D - x_i\right]$. This assumption and our definition of Sensitivity-Preprocessing Function together imply,

$$g(D - x_i) = \min_{j \neq i}\{g(D - x_i - x_j) + \Delta\}.$$

The definition of Sensitivity-Preprocessing Function also gives:

$$\min\{g(D-x_1), g(D-x_n)\} \leq \min\{\min_{j\neq 1}\{g(D-x_1-x_j)+\Delta\}, \min_{j\neq n}\{g(D-x_n-x_j)+\Delta\}\}.$$

As a result, if $\min_{j\neq 1}\{g(D-x_i-x_j)+\Delta\}$ is minimized for $j=1$ or $j=n$, then we easily have $\min\{g(D-x_1), g(D-x_n)\} \leq g(D-x_i)$. Furthermore, if $j \neq 1, n$, then it suffices to show that,

$$\min\{g(D-x_1-x_j), g(D-x_n-x_j)\} \leq g(D-x_i-x_j),$$

which follows from the inductive hypothesis and implies our desired result. □

These two helper lemmas now easily imply Lemma 4.6.4.

Proof of Lemma 4.6.4. Lemma 4.6.5 implies that we will never need to use $\textsc{Lower}(D)$, so we can eliminate that case. Further, Lemma 4.6.6 implies that $\textsc{Upper}(D) = \min\{g(D-x_1)+\Delta, g(D-x_n)+\Delta\}$. Combining these facts implies our recursion defined in the lemma statement is equivalent to the Sensitivity-Preprocessing Function. □

With this reduction in the number of subproblems for the Sensitivity-Preprocessing Function, we will be able to give a similar efficient dynamic programming algorithm for the implementation.

It immediately follows that the number of subproblems that we need to consider is $O(n^2)$, but we still need to efficiently compute **Var** $[D]$. This computation would normally take $O(n)$ time and increase our running time to $O(n^3)$. However, we can use the computation from previous subproblems to compute the variance in $O(1)$ time with the following folklore fact that we prove in the appendix.

Algorithm 12: Efficient Implementation for Variance

Input: Variance function **Var** : $\mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$, sensitivity bound Δ, and database $D = (x_1, ..., x_n) \in \mathbb{R}^n$ for some arbitrary n.
Output: $g(D)$ where g is the Sensitivity-Preprocessing Function of variance with parameter Δ.
Initialize $g(\emptyset) = 0$
Sort D (We will assume $x_1 \leq \cdots \leq x_n$ for simplicity) **for** *k=1, . . . , n* **do**
    for *i = 1, . . . n-k+1* **do**
        for *every database* $D' = (x_i, ..., x_{i+k-1})$ **do**
            Let $g(D') = \min\{\textbf{Var}\left[D'\right], g(D' - x_i) + \Delta, g(D' - x_{i+k-1}) + \Delta\}$
Output $g(D)$

Fact 4.6.7. *For any* $D = (x_1, ..., x_n) \in \mathbb{R}^n$ *and any* $x_a \neq x_b \in D$,

$$\textbf{Var}\left[D\right] = \left(\frac{n-1}{n}\right)^2 \textbf{Var}\left[D - x_a\right] + \left(\frac{n-1}{n}\right)^2 \textbf{Var}\left[D - x_b\right] - \left(\frac{n-2}{n}\right)^2 \textbf{Var}\left[D - x_a - x_b\right] + \frac{1}{n^2}(x_a - x_b)^2.$$

With this fact we can now show that we implement the Sensitivity-Preprocessing Function for variance with parameter Δ in $O(n^2)$ time.

Lemma 4.6.8. *Let* ***Var*** : $\mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ *be the variance function and set* $\textbf{Var}\left[\emptyset\right] = 0$. *Then Algorithm 12 will compute* $g(D)$ *for any database of* n *entries in* $O(n^2)$ *time where* g *is the Sensitivity-Preprocessing Function for variance with parameter* Δ.

Proof. Correctness of the procedure follows immediately from Lemma 4.6.4. The running time follows from the fact that we have $O(n^2)$ subproblems and from Fact 4.6.7 we can compute $\textbf{Var}\left[D\right]$ in $O(1)$ time using the previous subproblems. □

4.6.2 Accuracy guarantees for variance implementation

In this section we give stronger bounds on the error incurred by the Sensitivity-Preprocessing Function. The proofs will be similar to those in Section 4.5.3 for mean, but will be slightly

simpler due to that fact that the Sensitivity-Preprocessing Function will never go above the actual variance. As a result, we achieve a simpler form for the error of the Sensitivity-Preprocessing Function with respect to variance in the following lemma.

Lemma 4.6.9. *Given $D = (x_1, ..., x_n) \in \mathbb{R}^n$ and g that is the Sensitivity-Preprocessing Function of variance with parameter Δ and $g(\emptyset) = 0$, then there must exist some $D' \subseteq D$ such that $g(D) = \textbf{Var}[D'] + (n-k)\Delta$ for $k = |D'|$.*

Proof. We prove this inductively on the size of D and see immediately that the claim holds by construction for D with a single entry.

We then consider $D = (x_1, ..., x_n)$ and assume that our claim holds for all subsets. From Lemma 4.6.5 we know that $\textbf{Var}[D] \geq g(D)$ for all databases. If $\textbf{Var}[D] = g(D)$, then our claim is immediately implied. If $g(D) < \textbf{Var}[D]$ then we must have $g(D) = g(D - x_i) + \Delta$ for some x_i. Applying the inductive hypothesis on $g(D - x_i)$ gives our desired claim. □

With this lemma in hand, the main idea for bounding accuracy is to condition on the size of D', which we denote k, and give bounds separately for the cases when $k \leq n/2$ and $k \geq n/2$. When k is small we will just bound our error by $\textbf{Var}[D] - (n-k)\Delta$ and use the fact that $(n-k)\Delta$ is large. When k is large we will look at the loss in accuracy from $\textbf{Var}[D] - \textbf{Var}[D']$ where we will bound this by iteratively applying the following lemma.

Lemma 4.6.10. *For any $D = (x_1, ..., x_n) \in \mathbb{R}^n$ and any $x_a \in D$, then*

$$\textbf{Var}[D] - \textbf{Var}[D - x_a] \leq \frac{1}{n^2} \sum_{i=1}^{n} (x_a - x_i)^2 .$$

Proof. By the definition of variance,

$$\textbf{Var}[D] - \textbf{Var}[D - x_a] = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{1}{2} (x_i - x_j)^2 - \frac{1}{(n-1)^2} \sum_{i \neq a} \sum_{j \neq a} \frac{1}{2} (x_i - x_j)^2 .$$

This reduces to,

$$\textbf{Var}\left[D\right] - \textbf{Var}\left[D - x_a\right] = \frac{1}{n^2}\sum_{i=1}^{n}\left(x_a - x_i\right)^2 - \frac{2n-1}{n^2(n-1)^2}\sum_{i\neq a}\sum_{j\neq a}\frac{1}{2}\left(x_i - x_j\right)^2,$$

which gives our desired equality. □

Recall that we want to use this lemma to bound $\textbf{Var}\left[D\right] - \textbf{Var}\left[D'\right]$ where D' is a subset of D with size k. Suppose $D' = (x_1, ..., x_k)$ and let $D_i = (x_1, ..., x_i)$ for any i; we will use the fact that $\textbf{Var}\left[D\right] - \textbf{Var}\left[D'\right] = \sum_{i=k+1}^{n} \textbf{Var}\left[D_i\right] - \textbf{Var}\left[D_{i-1}\right]$. The above Lemma 4.6.10 allows us to bound this sum, which will be the key step in our accuracy bounds.

Lemma 4.6.11. *Given $D = (x_1, ..., x_n) \in \mathbb{R}^n$ and g that is the Sensitivity-Preprocessing Function of variance with parameter Δ and $g(\emptyset) = 0$, then*

$$\left|\textbf{Var}\left[D\right] - g(D)\right| \leq \max\left\{\textbf{Var}\left[D\right] - \frac{n}{2}\Delta, 0\right\} + \sum_{i=1}^{n}\max\left\{\sum_{j=1}^{n}\frac{4(x_i - x_j)^2}{n^2} - \Delta, 0\right\}.$$

Proof. Note that Lemma 4.6.5 implies $\left|\textbf{Var}\left[D\right] - g(D)\right| = \textbf{Var}\left[D\right] - g(D)$. From Lemma 4.6.9 we know that $g(D) = \textbf{Var}\left[D'\right] + (n-k)\Delta$ for some $D' \subseteq D$ of size k, and we can rewrite $\textbf{Var}\left[D\right] - g(D) = \textbf{Var}\left[D\right] - \textbf{Var}\left[D'\right] - (n-k)\Delta$. If $k \leq n/2$, then

$$\textbf{Var}\left[D\right] - g(D) \leq \textbf{Var}\left[D\right] - \frac{n}{2}\Delta,$$

because $(n-k) \geq n/2$ and $\textbf{Var}\left[D'\right] \geq 0$.

If $k \geq n/2$, then for simplicity we will assume $D' = (x_1, ..., x_k)$ and address this assumption later. We then let $D_i = (x_1, ..., x_i)$ for any i and use the fact that $\textbf{Var}\left[D\right] - \textbf{Var}\left[D'\right] = \sum_{i=k+1}^{n} \textbf{Var}\left[D_i\right] - \textbf{Var}\left[D_{i-1}\right]$. Lemma 4.6.10 along with the fact that $k \geq n/2$ allows us to then bound this summation as

$$\sum_{i=k+1}^{n} \textbf{Var}\left[D_i\right] - \textbf{Var}\left[D_{i-1}\right] \leq \frac{4}{n^2}\sum_{i=k+1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2$$

We can then use this to achieve (for $D' = (x_1, ..., x_k)$)

$$\textbf{Var}\left[D\right] - \textbf{Var}\left[D'\right] - (n-k)\Delta \leq \sum_{i=k+1}^{n}\left(\sum_{j=1}^{n}\frac{4(x_i - x_j)^2}{n^2} - \Delta\right)$$

At this point we address the assumption that $D' = (x_1, ..., x_k)$ by simply adding non-negative terms to the summation and ensuring that all of the entries in D' are be included in this summation. This gives us,

$$\textbf{Var}\left[D\right] - \textbf{Var}\left[D'\right] - (n-k)\Delta \leq \sum_{i=1}^{n}\max\left\{\sum_{j=1}^{n}\frac{4(x_i - x_j)^2}{n^2} - \Delta, 0\right\}.$$

Adding both errors for $k \leq n/2$ and $k \geq n/2$ gives our desired bound. □

4.6.3 Proof of Theorem 4.1.11

We now have all the necessary pieces for Theorem 4.1.11, which we restate and prove here.

Theorem 4.1.11. *Let* ***Var*** $: \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ *be the variance function for the data universe of all finite-length real-valued vectors. For fixed parameter* Δ*, along with any database* $D = (x_1, ..., x_n) \in \mathbb{R}^{<\mathbb{N}}$*, we have* $O(n^2)$ *time access to a function* $g : \mathbb{R}^{<\mathbb{N}} \to \mathbb{R}$ *with sensitivity* Δ *such that,*

$$|g(D) - \textbf{Var}\left[D\right]| \leq \max\left\{\textbf{Var}\left[D\right] - \frac{n}{2}\Delta, 0\right\} + \sum_{i=1}^{n}\max\left\{\sum_{j=1}^{n}\frac{4(x_i - x_j)^2}{n^2} - \Delta, 0\right\}.$$

Proof. The fact that g has sensitivity Δ follows from the fact that it is our Sensitivity-Preprocessing Function from Lemma 4.6.8, and the guarantees of Lemma 4.3.5. The runtime also follows from Lemma 4.6.8. We then achieve the error bounds from Lemma 4.6.11.

□

4.7 Sensitivity preprocessing for personalized privacy guarantees

In this section, we introduce *personalized differential privacy*, where each individual in a database may receive a different privacy parameter ϵ_i. We show that our Sensitivity-Preprocessing Function is naturally compatible with this privacy notion, and demonstrate the use of sensitivity-bounded functions for achieving personalized privacy guarantees, using the Laplace Mechanism and the Exponential Mechanism as illustrative examples. The notion of personalized privacy has been previously applied to the design of markets for privacy. We demonstrate the use of Sensitivity-Preprocessing Function for this application in Section 4.7.2, and hope that our results may be useful tools for this well-studied problem in algorithmic economics.

4.7.1 Personalized differential privacy

We begin by defining *personalized differential privacy*, which extends the standard definition of differential privacy (Definition 4.2.1) to a setting where different individuals participating in the same computation may experience different, personalized privacy guarantees. Similar definitions have also been used in previous work [176, 183, 177, 178]. Recall from Section 4.2 that two databases are neighboring if they differ in at most one entry. We will say that two databases are i-*neighbors* if they differ only in the i-th entry.

Definition 4.7.1 (Personalized differential privacy)**.** A mechanism $\mathcal{M} : \mathcal{D} \to \mathcal{R}$ is $\{\epsilon_i\}$-*personally differentially private* if for all i, for every pair of i-neighbors $D, D' \in \mathcal{D}$, and for every subset of possible outputs $\mathcal{S} \subseteq \mathcal{R}$,

$$\boldsymbol{Pr}[\mathcal{M}(D) \in \mathcal{S}] \leq \exp(\epsilon_i)\boldsymbol{Pr}[\mathcal{M}(D') \in \mathcal{S}].$$

Note that any $\{\epsilon_i\}$-personally differentially private algorithm is also $(\max_i \epsilon_i)$-differentially private, since differential privacy provides a worst-case guarantee over all pairs of neighboring databases.

In this section, we show that personalized differential privacy can be achieved by combining our sensitivity preprocessing step with existing differentially private mechanisms. An analyst can first apply our preprocessing step to get g with desired individual sensitivity bounds, and then evaluate g using a differentially private algorithm. The resulting $\{\epsilon_i\}$-personal differential privacy guarantees will depend on the chosen sensitivity parameters $\{\Delta_i\}$. Since the function g is independent of the database, the sensitivity preprocessing step does not leak any additional privacy.

Individual sensitivity guarantees are critical for accurate analysis in this new privacy model. Using only global sensitivity bounds Δ, personally differentially private mechanisms add noise that scales with $\max_i\{\Delta/\epsilon_i\}$. This alone cannot offer significant accuracy improvements because the noise must still scale inversely proportionally to the smallest ϵ_i. By utilizing individual sensitivity bounds, an analyst can tune each Δ_i to scale with ϵ_i to achieve overall accuracy improvements with personalized differential privacy.

We note that *local differential privacy* [184] also affords different privacy guarantees to different individuals in the same database, by perturbing each user's data locally before submitting it to the database. Significantly stronger accuracy guarantees are possible in the presence of a trusted curator—which we assume in our model—because the analyst can leverage correlation of noise across individuals [185].

A formal statement of the privacy and accuracy guarantees that arise from applying differentially private algorithms to sensitivity-bounded functions will depend on the exact algorithm used. We illustrate this approach below applying it on two of the most foundational differentially private algorithms: the Laplace Mechanism and the Exponential Mechanism.

Laplace Mechanism

The *Laplace Mechanism* [161] is perhaps the most fundamental of all differentially private algorithms. It first evaluates a real-valued function f on an input database D, and then perturbs the answer by adding Laplace noise scaled to the global sensitivity of f divided by

ϵ. The *Laplace distribution* with scale b, denoted $\mathrm{Lap}(b)$, has probability density function:

$$\mathrm{Lap}(x|b) = \frac{1}{2b} \exp\left(-\frac{|x|}{b}\right).$$

Definition 4.7.2 (Laplace Mechanism [161])**.** Given any function $f : \mathcal{D} \to \mathbb{R}$, the *Laplace Mechanism* is defined as,

$$\mathcal{M}_L(D, f, \Delta f/\epsilon) = f(D) + Y,$$

where Y is drawn from $\mathrm{Lap}(\Delta f/\epsilon)$.[6]

The Laplace Mechanism is ϵ-differentially private [161]. We now show how to combine the Laplace Mechanism with our Sensitivity-Preprocessing Function to achieve personalized differential privacy guarantees.

Proposition 4.7.3. *Let $g : \mathcal{D} \to \mathbb{R}$ be a function with individual sensitivities $\{\Delta_i\}$. For any $\{\epsilon_i\}$, the Laplace Mechanism $\mathcal{M}_L\left(D, g, \max_j\{\Delta_j/\epsilon_j\}\right)$ is $\{\epsilon_i\}$-personally differentially private.*

Proof. Let $D, D' \in \mathcal{D}$ be i-neighbors, let $g : \mathcal{D} \to \mathbb{R}$ be a function with individual

[6]We note that the standard definition of the Laplace Mechanism in [161] takes ϵ as input instead of $\frac{\Delta f}{\epsilon}$. We use the latter here for ease of notation when extending to personalized differential privacy. This change does not affect the algorithm at all.

sensitivities $\{\Delta_i\}$, and let $r \in \mathbb{R}$ be arbitrary.

$$\begin{aligned}
\frac{\boldsymbol{Pr}[\mathcal{M}_L(D, g, \max_j\{\Delta_j/\epsilon_j\}) = r]}{\boldsymbol{Pr}[\mathcal{M}_L(D', g, \max_j\{\Delta_j/\epsilon_j\}) = r]} &= \frac{\exp\left(-\max_j\{\frac{\epsilon_j}{\Delta_j}\}|g(D) - r|\right)}{\exp\left(-\max_j\{\frac{\epsilon_j}{\Delta_j}\}|g(D') - r|\right)} \\
&= \exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}\left(|g(D') - r| - |g(D) - r|\right)\right) \\
&\leq \exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}\left(|g(D) - g(D')|\right)\right) \\
&\leq \exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}\Delta_i\right) \\
&\leq \exp(\epsilon_i)
\end{aligned}$$

Then this version of the Laplace Mechanism run on a function with individual sensitivities $\{\Delta_i\}$ is $\{\epsilon_i\}$-personally differentially private. □

Proposition 4.7.3 shows that to achieve personalized privacy guarantees for a given function f, one can apply our Sensitivity-Preprocessing Function to produce Sensitivity-Bounded g, and then apply the Laplace Mechanism. The accuracy guarantees of this procedure will depend on the worst-case ratio of Δ_i/ϵ_i, as well as global sensitivity of the original function f. If one person j requires significantly higher privacy protections than the rest of the population, the analyst can account for this by reducing Δ_j. This may greatly improve accuracy over the standard approach, which would require the analyst to add increased noise to the entire population. We address this challenge more concretely in Section 4.7.2, using the application of market design for private data.

Exponential Mechanism

The *Exponential Mechanism* [186] is a powerful private mechanism for answering non-numeric queries with an arbitrary range, such as selecting the best outcome from a set of alternatives. The quality of an outcome is measured by a *score function* $q\colon \mathcal{D} \times \mathcal{R} \to \mathbb{R}$, which relates each alternative to the underlying data through a real-valued score. The global

sensitivity of the score function is measured only with respect to the database argument; it can be arbitrarily sensitive in its range argument:

$$\Delta q = \max_{r \in \mathcal{R}} \max_{D, D' \; neighbors} |q(D, r) - q(D', r)|.$$

We define the individual sensitivity of a quality score analogously with respect to only its database argument:

$$\Delta_i(q) = \max_{r \in \mathcal{R}} \max_{D, D' \; i-neighbors} |q(D, r) - q(D', r)|.$$

The Exponential Mechanism samples an output from the range $\mathcal{R}$ with probability exponentially weighted by score. Outcomes with higher scores are exponentially more likely to be selected, thus ensuring both privacy and a high quality outcome.

Definition 4.7.4 (Exponential Mechanism [186])**.** Given a quality score $q : \mathcal{D} \times \mathcal{R} \to \mathbb{R}$, the *Exponential Mechanism* is defined as:[7]

$$\mathcal{M}_E(D, q, \Delta q/\epsilon) = \text{output } r \in \mathcal{R} \text{ with probability proportional to } \exp\left(\frac{\epsilon q(D, r)}{2\Delta q}\right).$$

The Exponential Mechanism is ϵ-differentially private [186]. We now show that when a score function has bounded individual sensitivity, the Exponential Mechanism is personally differentially private.

Proposition 4.7.5. *Let $q : \mathcal{D} \times \mathcal{R} \to \mathbb{R}$ be a score function with individual sensitivities $\{\Delta_i\}$. For any $\{\epsilon_i\}$, the Exponential Mechanism $\mathcal{M}_E\left(D, q, \max_j\{\Delta_j/\epsilon_j\}\right)$ is $\{\epsilon_i\}$-personally differentially private.*

Proof. Let $D, D' \in \mathcal{D}$ be i-neighbors, let q be a score function with individual sensitivities

[7] As with the Laplace Mechanism, we define the Exponential Mechanism to take $\frac{\Delta q}{\epsilon}$ as input, instead of ϵ. This change is purely notational, and has no impact on the algorithm.

$\{\Delta_i\}$, and let $r \in \mathcal{R}$ be an arbitrary element of the output range.

$$
\begin{aligned}
&\frac{\boldsymbol{Pr}[\mathcal{M}_E(D, q, \max_j\{\Delta_j/\epsilon_j\}) = r]}{\boldsymbol{Pr}[\mathcal{M}_E(D', q, \max_j\{\Delta_j/\epsilon_j\}) = r]} \\
&\quad = \frac{\left(\frac{\exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}q(D,r)/2\right)}{\sum_{r'\in\mathcal{R}}\exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}q(D,r')/2\right)}\right)}{\left(\frac{\exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}q(D',r)/2\right)}{\sum_{r'\in\mathcal{R}}\exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}q(D',r')/2\right)}\right)} \\
&\quad = \left(\frac{\exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}q(D,r)/2\right)}{\exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}q(D',r)/2\right)}\right) \cdot \left(\frac{\sum_{r'\in\mathcal{R}}\exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}q(D',r')/2\right)}{\sum_{r'\in\mathcal{R}}\exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}q(D,r')/2\right)}\right) \\
&\quad = \exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}\left(q(D,r) - q(D',r)\right)/2\right) \cdot \left(\frac{\sum_{r'\in\mathcal{R}}\exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}q(D',r')/2\right)}{\sum_{r'\in\mathcal{R}}\exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}q(D,r')/2\right)}\right) \\
&\quad \leq \exp\left(\frac{1}{2}\max_j\{\frac{\epsilon_j}{\Delta_j}\}\Delta_i\right) \cdot \exp\left(\frac{1}{2}\max_j\{\frac{\epsilon_j}{\Delta_j}\}\Delta_i\right) \cdot \left(\frac{\sum_{r'\in\mathcal{R}}\exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}q(D,r')/2\right)}{\sum_{r'\in\mathcal{R}}\exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}q(D,r')/2\right)}\right) \\
&\quad = \exp\left(\max_j\{\frac{\epsilon_j}{\Delta_j}\}\Delta_i\right) \\
&\quad \leq \exp(\epsilon_i)
\end{aligned}
$$

□

Remark 4.7.6. The Exponential Mechanism is a canonical ϵ-differentially private algorithm: every ϵ-differentially private algorithm $\mathcal{M}$ can be written as an instantiation of the Exponential Mechanism using quality score $q(D, r) = \ln(\boldsymbol{Pr}[\mathcal{M}(D) = r])$ with global sensitivity $\Delta q = \epsilon$. We can use this reduction to show that *any* ϵ-differentially private algorithm can be modified to give personal privacy guarantees using our Sensitivity-Preprocessing Function. First, re-write private mechanism $\mathcal{M}$ as an Exponential Mechanism $\mathcal{M}_E$, and then perform Sensitivity-Preprocessing on the quality score of $\mathcal{M}_E$. Proposition 4.7.5 shows that the sensitivity-bounded version of $\mathcal{M}_E$ will satisfy personalized differential privacy.

4.7.2 Application: Markets for privacy

One motivating application for wanting personalized privacy guarantees come from algorithmic game theory and the study of market design for privacy. This is a well-studied problem in the algorithmic economics community [166, 168, 169, 170, 171, 165, 172, 167, 173, 174, 175], and of practical importance as growing amounts of data are collected about individuals. In a *market for privacy*, a data analyst wishes to purchase and aggregate data from multiple strategic individuals. These individuals may have privacy concerns, and will require compensation for their privacy loss from sharing data. On the opposite side of the market, firms demand accurate estimates of population statistics, for uses such as market research or operational decision making.

The analyst must first purchase data from these strategic individuals, and then aggregate the collected data into an accurate estimate for firms. Her goal is to perform this task while maximizing her own profits. One of the tools at her disposal is differential privacy: by offering individuals formal privacy guarantees, their privacy costs from sharing data are diminished, and the analyst can provide smaller payments. However, the noise from differential privacy may introduce additional error.

It is the analyst's task to determine the optimal privacy level for the market that balances these opposing effects. Due to potentially heterogeneous privacy costs of the individuals, it may be optimal in terms of her profit for the analyst to provide different privacy guarantees to different individuals in the population. She could then use our Sensitivity-Preprocessing Function to algorithmically provide the heterogeneous privacy levels demanded by the market. We leave the challenge of modeling specifics of these markets as an open question to the algorithmic game theory community, and hope our preprocessing tool and mechanisms for personalized privacy will open new avenues for designing markets for privacy.

4.8 Extension to 2-dimensions for ℓ_1 sensitivity

In this section we show that our Sensitivity-Preprocessing Function can be naturally extended to functions that map to 2-dimensional space where we consider the sensitivity in the ℓ_1 distance metric. While there is a natural extension of our Sensitivity-Preprocessing Function to higher dimensions, the primary difficulty will be ensuring that our greedy construction still yields a non-empty intersection of the constraints. Interestingly, we show that this set of constraints will give a non-empty intersection for 2 dimensions, and provide a counter-example for higher dimensions.

Recall that our Sensitivity-Preprocessing Function found a range $[\textsc{Lower}(D), \textsc{Upper}(D)]$ where it could feasibly place $g(D)$, then choose the point in that segment closest to $f(D)$. This range of feasible points came from intersecting each constraint $[g(D - x_i) - \Delta_i, g(D - x_i) + \Delta_i]$ induced by the neighbors of D that are strictly smaller. The Sensitivity-Preprocessing Function then chose the point in this intersection closest to $f(D)$. The key property needed by the algorithm was that this intersection was non-empty.

To prove this key property we took advantage of the data universe structure, which immediately yielded the fact that for any $x_i, x_j \in D$ we must have $[g(D - x_i) - \Delta_i, g(D - x_i) + \Delta_i] \cap [g(D - x_j) - \Delta_j, g(D - x_j) + \Delta_j] \neq \emptyset$. As a result, we had a finite set of line segments whose intersection was pair-wise non-empty, which immediately implies that the intersection of all line segments was non-empty. We note that $[g(D-x_i)-\Delta_i, g(D-x_i)+\Delta_i]$ is the ℓ_1 ball with radius Δ_i around $g(D - x_i)$ in one dimension. For higher dimensions, the constraints will now be the ℓ_1 ball with radius Δ_i around $g(D - x_i)$ in d dimensions. The structure of the data universe will still give that each of these ℓ_1 balls has a non-empty pair-wise intersection. However, this only implies that the intersection of all these ℓ_1 balls is non-empty if we are in 2 dimensions. We first formally define the notion of an ℓ_1 ball in higher dimensions.

Definition 4.8.1 (ℓ_1 ball)**.** The ℓ_1 ball around point $x^* \in \mathbb{R}^d$ with radius Δ is the set:

$$(x^*, \Delta)_1^d \stackrel{\text{def}}{=} \{x \in \mathbb{R}^d | \, \|x - x^*\|_1 \leq \Delta\}.$$

To ensure that our choice of $g(D)$ does not violate the individual sensitivity parameter Δ_i, we must place $g(D) \in (g(D - x_i), \Delta_i)_1^d$. In the one-dimensional case, we had the same constraints, but they were simpler to handle because the ℓ_1 ball is simply a line segment. We now define our Sensitivity-Preprocessing Function for two dimensions, which chooses the point that satisfies our constraints and is closest to $f(D)$, just as in the one-dimensional case.

Definition 4.8.2 (2-dimensional Sensitivity-Preprocessing Function)**.** Given a function $f : \mathcal{D} \to \mathbb{R}^2$ for any data universe such that for any $D \in \mathcal{D}$, all $D' \subset D$ are also in $\mathcal{D}$. For any non-negative individual sensitivity parameters $\{\Delta_i\}$, we say that a function $g : \mathcal{D} \to \mathbb{R}^2$ is a Sensitivity-Preprocessing Function of f with parameters $\{\Delta_i\}$ if $g(\emptyset) = f(\emptyset)$ and

$$g(D) = \text{closest point in } \cap_{x_i \in D} \, (g(D - x_i), \Delta_i)_1^2 \text{ to } f(D) \text{ in the } \ell_2 \text{ metric.}$$

If all $\Delta_i = \Delta$ for some non-negative Δ, then we say that g is a Sensitivity-Preprocessing Function of f with parameter Δ.

Our primary goal of this section will then be to prove the following theorem that is equivalent to Theorem 4.3.3 but works for 2-dimensions. We will also point out the key spot within the proof where it breaks for dimensions greater than 2.

Theorem 4.8.3. *Given $T(n)$-time query access to an arbitrary $f : \mathcal{D} \to \mathbb{R}^2$, and sensitivity parameters $\{\Delta_i\}$, we provide $O((T(n) + n)2^n)$ time access to Sensitivity-Preprocessing Function $g : \mathcal{D} \to \mathbb{R}^2$ such that $\Delta_i(g) \leq \Delta_i$. Further, for any database $D = (x_1, \ldots, x_n)$,*

$$\|f(D) - g(D)\|_1 \leq \max_{\sigma \in \sigma_D} \sum_{i=1}^{|D|} \max\{\left\|f(D_{\sigma(<i)} + x_{\sigma(i)}) - f(D_{\sigma(<i)})\right\|_1 - \Delta_{\sigma(i)}, 0\},$$

where σ_D is the set of all permutations on $[n]$, and $D_{\sigma(<i)} = (x_{\sigma(1)}, ..., x_{\sigma(i-1)})$ is the subset of D that includes all individual data in the permutation before the ith entry.

As before, we will break the proof of this theorem into two parts. It immediately follows from construction that our 2-dimensional Sensitivity-Preprocessing Function will have the appropriate individual sensitivity parameters, but only if the function is well-defined. To this end, we first show in Section 4.8.1 that the intersection of the ℓ_1 balls is always non-empty if each pair-wise intersection is non-empty. Then in Section 4.8.2 we give the analogous error guarantees where the proof will just follow equivalently to the one-dimensional case.

4.8.1 Correctness of Sensitivity-Preprocessing Function

In this section we show that for our 2-dimensional Sensitivity-Preprocessing Function, it is always the case that $g(D)$ is defined. This is equivalent to showing:

$$\bigcap_{x_i \in D} (g(D - x_i), \Delta_i)_1^2 \neq \emptyset.$$

We will first take advantage of the structure of data universes to show that the pair-wise intersection is always non-empty. Then we will use the fact that pair-wise intersection of ℓ_1 balls in 2-dimensions implies that the intersection of all ℓ_1 balls is non-empty. Intuitively, this is because ℓ_1 balls in 2-dimensions are simply rotated squares. Further, we will show that this is exactly the step that breaks the algorithm for higher dimensions.

Lemma 4.8.4. *Given any $f : \mathcal{D} \to \mathbb{R}^2$ and desired sensitivity parameters $\{\Delta_i\}$, let $g : \mathcal{D} \to \mathbb{R}^2$ be the Sensitivity-Preprocessing Function with parameters $\{\Delta_i\}$. For any $D \in \mathcal{D}$ with at least two entries, assume that $g(D')$ is defined for any $D' \subset D$. Then for any $x_i, x_j \in D$,*

$$(g(D - x_i), \Delta_i)_1^2 \cap (g(D - x_j), \Delta_j)_1^2 \neq \emptyset.$$

Note that we have not yet proven that $g(D)$ is defined on all databases, so we will need to first assume that it is on all subsets of D. Our proof of this fact will be done inductively.

Proof. We use the fact that D has at least two entries and consider the database $D - x_i - x_j$. Due to our assumption that $g(D')$ is defined on all $D' \subset D$, it follows from our construction of g that

$$\|g(D - x_i) - g(D - x_i - x_j)\|_1 \leq \Delta_j,$$

and

$$\|g(D - x_j) - g(D - x_i - x_j)\|_1 \leq \Delta_i,$$

Applying triangle inequality gives,

$$\|g(D - x_i) - g(D - x_j)\|_1 \leq \Delta_i + \Delta_j,$$

which implies our claim by the definition of ℓ_1 balls. □

With this pair-wise intersection property, it now remains to be shown that this implies the intersection of all ℓ_1 balls is non-empty. For this we prove a general fact about the intersection of ℓ_1 balls in 2-dimensions.

Lemma 4.8.5. *Consider any set of points $y_1, ..., y_n \in \mathbb{R}^2$, where we let $(y_i)_1$ and $(y_i)_2$ denote the respective coordinates of y_i. Consider any set of non-negative $\Delta_1, ..., \Delta_n$. If for any y_i, y_j,*

$$(y_i, \Delta_i)_1^2 \cap (y_j, \Delta_j)_1^2 \neq \emptyset,$$

then,

$$\bigcap_{i=1}^{n} (y_i, \Delta_i)_1^2 \neq \emptyset.$$

Our proof will first rewrite each ℓ_1 ball as a set of 4 linear inequalities. From this interpretation we will then use two critical facts. First, each inequality has a corresponding parallel inequality in any other ℓ_1 ball. Second, removing any one of these constraints gives an unbounded polytope.

Proof. Further examination of Definition 4.8.1 shows that

$$(y_i, \Delta_i)_1^2 \stackrel{\text{def}}{=} \{x \in \mathbb{R}^2 |\, \|x - y_i\|_1 \leq \Delta_i\} = \{x \in \mathbb{R}^2 |\, |(x)_1 - (y_i)_1| + |(x)_2 - (y_i)_2| \leq \Delta_i\}.$$

We then use a known trick of converting absolute values into linear inequalities where $|x| \leq k$ becomes $x \leq k$ and $-x \leq k$.

$$\begin{aligned}(y_i, \Delta_i)_1^2 =& \{x \in \mathbb{R}^2 | (x)_1 + (x)_2 \leq (y_i)_1 + (y_i)_2 + \Delta_i\} \\ & \cap \{x \in \mathbb{R}^2 | -(x)_1 - (x)_2 \leq -(y_i)_1 - (y_i)_2 + \Delta_i\} \\ & \cap \{x \in \mathbb{R}^2 | (x)_1 - (x)_2 \leq (y_i)_1 - (y_i)_2 + \Delta_i\} \\ & \cap \{x \in \mathbb{R}^2 | -(x)_1 + (x)_2 \leq -(y_i)_1 + (y_i)_2 + \Delta_i\}\end{aligned}$$

At this point we note that each of the balls have parallel inequalities, so we can use the following fact:

$$\begin{aligned}&\{x \in \mathbb{R}^2 | (x)_1 + (x)_2 \leq (y_i)_1 + (y_i)_2 + \Delta_i\} \cap \{x \in \mathbb{R}^2 | (x)_1 + (x)_2 \leq (y_j)_1 + (y_j)_2 + \Delta_j\} \\ &\quad = \left\{x \in \mathbb{R}^2 | (x)_1 + (x)_2 \leq \min\{(y_i)_1 + (y_i)_2 + \Delta_i, (y_j)_1 + (y_j)_2 + \Delta_j\}\right\}.\end{aligned}$$

We apply this fact to the full intersection and obtain,

$$\begin{aligned}\bigcap_{i=1}^{n} (y_i, \Delta_i)_1^2 =& \{x \in \mathbb{R}^2 | (x)_1 + (x)_2 \leq \min_{i \in [n]} \{(y_i)_1 + (y_i)_2 + \Delta_i\}\} \\ & \cap \{x \in \mathbb{R}^2 | -(x)_1 - (x)_2 \leq \min_{i \in [n]} \{-(y_i)_1 - (y_i)_2 + \Delta_i\}\} \\ & \cap \{x \in \mathbb{R}^2 | (x)_1 - (x)_2 \leq \min_{i \in [n]} \{(y_i)_1 - (y_i)_2 + \Delta_i\}\} \\ & \cap \{x \in \mathbb{R}^2 | -(x)_1 + (x)_2 \leq \min_{i \in [n]} \{-(y_i)_1 + (y_i)_2 + \Delta_i\}\}.\end{aligned}$$

Intuitively, if this intersection exists, it must be a rectangle that is rotated 45 degrees. To

more easily see this fact, we now multiply the second and fourth constraint by -1.

$$\begin{aligned}\bigcap_{i=1}^{n}(y_i,\Delta_i)_1^2 =&\{x\in\mathbb{R}^2|(x)_1+(x)_2\leq\min_{i\in[n]}\{(y_i)_1+(y_i)_2+\Delta_i\}\}\\ &\cap\{x\in\mathbb{R}^2|(x)_1+(x)_2\geq\min_{i\in[n]}\{(y_i)_1+(y_i)_2-\Delta_i\}\}\\ &\cap\{x\in\mathbb{R}^2|(x)_1-(x)_2\leq\min_{i\in[n]}\{(y_i)_1-(y_i)_2+\Delta_i\}\}\\ &\cap\{x\in\mathbb{R}^2|(x)_1-(x)_2\geq\min_{i\in[n]}\{(y_i)_1-(y_i)_2-\Delta_i\}\}\end{aligned}$$

With this interpretation it is straightforward to see that $\bigcap_{i=1}^{n}(y_i,\Delta_i)_1^2=\emptyset$ if and only if

$$\min_{i\in[n]}\{(y_i)_1+(y_i)_2+\Delta_i\}<\min_{i\in[n]}\{(y_i)_1+(y_i)_2-\Delta_i\},$$

or

$$\min_{i\in[n]}\{(y_i)_1-(y_i)_2+\Delta_i\}<\min_{i\in[n]}\{(y_i)_1-(y_i)_2-\Delta_i\}.$$

Let k be the index that minimizes $(y_i)_1+(y_i)_2+\Delta_i$ and let l be the index that minimizes $(y_i)_1+(y_i)_2-\Delta_i$. If

$$(y_k)_1+(y_k)_2+\Delta_k<(y_l)_1+(y_l)_2-\Delta_l,$$

then we must have,

$$\Delta_k+\Delta_l<(y_l)_1-(y_k)_1+(y_l)_2-(y_k)_2\leq|(y_l)_1-(y_k)_1|+|(y_l)_2-(y_k)_2|\,,$$

which contradicts our assumption that $(y_k,\Delta_k)_1^2\cap(y_l,\Delta_l)_1^2\neq\emptyset$. This follows identically for the second inequality, so therefore neither of them can hold and the intersection must be non-empty. □

We now remark that Theorem 4.8.3 cannot be extended to higher dimensions or to ℓ_p norms.

Remark 4.8.6. For extending to dimensions greater than two, the proof breaks down at Lemma 4.8.5. Intuitively, we can still interpret each ℓ_1 ball as a set of linear inequalities, however it no longer has the critical property that removing one of the constraints creates an unbounded polytope. More specifically, consider the following counter-example for 3 dimensions:

Let $A = \{(1,1,-1),(1,-1,1),(-1,1,1),(1,-1,-1),(-1,1,-1),(-1,-1,1)\}$ and set $\Delta = 3$. It is not difficult to see that taking the intersection of Δ-radius ℓ_1 balls around each point in A will only contain the origin $(0,0,0)$. We then consider adding the point $(3/2,3/2,3/2)$, and it is straightforward to verify that the ℓ_1 distance between this point and any point in A is at most $11/2 < 2\Delta$. However, the origin is not within the ℓ_1 ball around $(3/2,3/2,3/2)$. Therefore, if we consider the set of ℓ_1 balls of radius Δ around the points in $A \cup (3/2,3/2,3/2)$, then each pair of ℓ_1 balls will intersect, but the full intersection will be empty, giving our counter-example.

Remark 4.8.7. Even in 2-dimensions, we cannot have Lemma 4.8.5 for the ℓ_p ball with $p \in (1,\infty)$ due to the curvature of each ball. For instance, consider the ℓ_2 ball with radius 1 for the points $(-1,0),(1,0),(0,\sqrt{3})$. Each of pair of these points is exactly distance 2 apart in the ℓ_2 metric, so their ℓ_2 balls of radius 1 each pairwise intersect. However it is easy to see that the intersection of all three is empty.

We can similarly extend this counter-example to other ℓ_p balls using the fact that there must be some curvature of the ℓ_p ball, and the midpoint between any two points in the ℓ_p metric is unique if $p \in (1,\infty)$.

With these lemmas, we are now able to show that our 2-dimensional Sensitivity-Preprocessing Function must always be defined.

Lemma 4.8.8. *Given any $f : \mathcal{D} \to \mathbb{R}^2$ with sensitivity parameters $\{\Delta_i\}$, let $g : \mathcal{D} \to \mathbb{R}^2$ be*

the Sensitivity-Preprocessing Function with parameters $\{\Delta_i\}$*. Then for any* $D \in \mathcal{D}$*,*

$$\bigcap_{x_i \in D} (g(D - x_i), \Delta_i)_1^2 \neq \emptyset.$$

Proof. We will prove this fact inductively, and note that it is immediately true when D only has one entry.

We then consider an arbitrary database D and assume that it is true for all $D' \subset D$. With this inductive claim we can apply Lemma 4.8.4 to get that all of the ℓ_1 balls have non-empty pairwise intersection. Our desired result then immediately follows from applying Lemma 4.8.5. □

4.8.2 Error bounds for the 2-dimensional extension

The following lemma gives the desired error bounds on the 2-dimensional Sensitivity-Preprocessing Function.

Lemma 4.8.9. *Given any* $f : \mathcal{D} \to \mathbb{R}^2$ *and desired sensitivity parameters* $\{\Delta_i\}$*, let* $g : \mathcal{D} \to \mathbb{R}^2$ *be the Sensitivity-Preprocessing Function with parameters* $\{\Delta_i\}$*. Then for any* $D \in \mathcal{D}$*,*

$$\|f(D) - g(D)\|_1 \leq \max_{\sigma \in \sigma_D} \sum_{i=1}^{|D|} \max\{\left\|f(D_{\sigma(<i)} + x_{\sigma(i)}) - f(D_{\sigma(<i)})\right\|_1 - \Delta_{\sigma(i)}, 0\},$$

where σ_D *is the set of all permutations of the set* $[n]$*, and let* $D_{\sigma(<i)} = (x_{\sigma(1)}, ..., x_{\sigma(i-1)})$ *be the subset of* D *that includes all individual data in the permutation before the* i*th entry.*

The proof of Lemma 4.8.9 follows identically to the proof of Lemma 4.3.6 where by replacing any instance of absolute value with the 1-norm.

We are finally ready to complete the proof of our main theorem for two dimensions.

Proof of Theorem 4.8.3. The individual sensitivity guarantees follow from the construction of g and Lemma 4.8.8. The error bounds are given by Lemma 4.8.9. It then remains to

prove the running time. For each subset of D we need to query f which takes $T(n)$ time by assumption. We note that within the proof of Lemma 4.8.5 we gave a construction for obtaining the intersection of n different ℓ_1 balls which could clearly be done in $O(n)$ time. Finding the closest point to $f(D)$ then takes $O(1)$ time for the polytope defined by four inequalities. Therefore, the running time is $T(n) + O(n)$ for each of the 2^n subsets, which implies the desired running time. □

4.9 Future Directions

We are especially interested in efficiently implementing our framework for more complicated and, in particular, higher-dimensional functions such as linear regression. We believe that leveraging the simple recursive construction of our algorithm along with non-trivial structural properties of these more difficult functions can allow for efficient and accurate implementation. We are particularly optimistic because all of our proofs in this work were from first principles, suggesting that we may be able to obtain further results from this framework by using more sophisticated tools.

While our construction did not generalize to any dimension under the ℓ_1 sensitivity metric, we note that this was in the most general setting. If the class of functions we consider is significantly restricted, then we believe the natural extension could both work and be efficiently implementable. Furthermore, we have not yet investigated variants of our algorithm that might work better under stronger assumptions or combining our construction with other frameworks for handling worst-case sensitivity.

We also believe that our construction opens up several intriguing directions with respect to personalized differential privacy and its application in markets for privacy. Our construction allows for tailoring individual sensitivity, but this presents a natural trade-off between choosing small individual sensitivity parameters and the error incurred by our preprocessing step. For specific functions, this may yield interesting optimization problems that can also be considered in the context of markets for privacy.

4.10 Omitted Proofs

In this appendix we provide proofs that were omitted from Sections 4.5 and 4.6.

4.10.1 Proof of Lemma 4.5.12

Proof of Lemma 4.5.12. We start by decomposing the RHS:

$$\frac{1}{n}\sum_{i=k}^{n}|x_i-\mu_D| = \frac{1}{n}\sum_{i=k}^{n}\left|x_i-\frac{x_1+\cdots+x_n}{n}\right| = \frac{1}{n^2}\sum_{i=k}^{n}\left|\sum_{j=1}^{n}(x_i-x_j)\right|.$$

It now suffices to show,

$$\sum_{i=k}^{n}\sum_{j=1}^{i}\frac{1}{3}|x_i-x_j| \leq \sum_{i=k}^{n}\left|\sum_{j=1}^{n}(x_i-x_j)\right|.$$

We further examine the RHS and use our assumption that $x_1 \leq \cdots \leq x_n$, which implies that for each i,

$$\left|\sum_{j=1}^{n}(x_i-x_j)\right| = \max\{\sum_{j=1}^{i}|x_i-x_j| - \sum_{j=i}^{n}|x_i-x_j|\,, \sum_{j=i}^{n}|x_i-x_j| - \sum_{j=1}^{i}|x_i-x_j|\}.$$

The idea will then be that because we have an ordering on $x_1, ..., x_n$, there will be a transition index. In particular, there is some $l \in [n-1]$ such that

$$\left|\sum_{j=1}^{n}(x_i-x_j)\right| = \sum_{j=1}^{i}|x_i-x_j| - \sum_{j=i}^{n}|x_i-x_j|$$

for all $i > l$, and

$$\left|\sum_{j=1}^{n}(x_i-x_j)\right| = \sum_{j=i}^{n}|x_i-x_j| - \sum_{j=1}^{i}|x_i-x_j|$$

for all $i \leq l$. If we then have $l \leq k$, then,

$$\sum_{i=k}^{n}\left|\sum_{j=1}^{n}(x_i-x_j)\right| = \sum_{i=k}^{n}\left(\sum_{j=1}^{i}|x_i-x_j| - \sum_{j=i}^{n}|x_i-x_j|\right).$$

By cancellation, we get,

$$\sum_{i=k}^{n}\left|\sum_{j=1}^{n}(x_i - x_j)\right| = \sum_{i=k}^{n}\sum_{j=1}^{k}|x_i - x_j|\,.$$

Applying Fact 4.10.1 gives,

$$2\sum_{i=k}^{n}\sum_{j=1}^{k}|x_i - x_j| \geq \sum_{i=k}^{n}\sum_{j=1}^{n}|x_i - x_j| \geq \sum_{i=k}^{n}\sum_{j=1}^{i}|x_i - x_j|\,,$$

as desired. If $l > k$, then,

$$\sum_{i=k}^{n}\left|\sum_{j=1}^{n}(x_i - x_j)\right| = \sum_{i=k}^{l}\left(\sum_{j=i}^{n}|x_i - x_j| - \sum_{j=1}^{i}|x_i - x_j|\right) + \sum_{i=l+1}^{n}\left(\sum_{j=1}^{i}|x_i - x_j| - \sum_{j=i}^{n}|x_i - x_j|\right).$$

We will simply lower bound the first term in the sum by 0, and note that Fact 4.10.1 (stated below) implies,

$$\sum_{i=k}^{l}\sum_{j=l}^{n}|x_i - x_j| \geq \sum_{i=k}^{l}\sum_{j=1}^{i}|x_i - x_j|\,.$$

Furthermore, by cancellation, we get,

$$\sum_{i=k}^{n}\left|\sum_{j=1}^{n}(x_i - x_j)\right| \geq \sum_{i=l+1}^{n}\sum_{j=1}^{l+1}|x_i - x_j|\,.$$

We then use the fact that,

$$\sum_{i=l+1}^{n}\sum_{j=1}^{l+1}|x_i - x_j| \geq \sum_{i=k}^{l}\sum_{j=1}^{i}|x_i - x_j|\,,$$

and

$$\sum_{i=l+1}^{n}\sum_{j=1}^{l+1}|x_i - x_j| \geq \sum_{i=l+1}^{n}\sum_{j=l}^{i}|x_i - x + j|\,,$$

from Fact 4.10.1, to obtain:

$$3\sum_{i=l+1}^{n}\sum_{j=1}^{l+1}|x_i - x_j| \geq \sum_{i=k}^{n}\sum_{j=1}^{i}|x_i - x_j|,$$

as desired. □

Fact 4.10.1. *For any ordered values $x_1 \leq \cdots \leq x_n$, and any $k \in [n-1]$ such that,*

$$\sum_{j=1}^{k}|x_k - x_j| \leq \sum_{j=k+1}^{n}|x_k - x_j|,$$

then for any $i \leq k$, we must have,

$$\sum_{j=1}^{k}|x_i - x_j| \leq \sum_{j=k+1}^{n}|x_i - x_j|.$$

Proof. This follows from the fact that $x_1 \leq \cdots \leq x_k$, so $\sum_{j=1}^{k}|x_i - x_j| \leq \sum_{j=1}^{k}|x_k - x_j|$ and $\sum_{j=k+1}^{n}|x_k - x_j| \leq \sum_{j=k+1}^{n}|x_i - x_j|$. □

4.10.2 Omitted proofs from Section 4.6

In this section we prove some important facts about variance that were necessary for obtaining an efficient algorithm for variance. We first show the intuitive fact that if we want to decrease the variance most, we should remove the maximum or minimum value.

Proof of Fact 4.6.2. It suffices to show that $\textbf{Var}[D - x_1] \leq \textbf{Var}[D - x_i]$ if $x_i \leq \mu(D)$ and that $\textbf{Var}[D - x_n] \leq \textbf{Var}[D - x_i]$ if $x_i \geq \mu(D)$. We will show the first, and the second follows equivalently.

We again use the definition of variance stated as,

$$\textbf{Var}[x_1, ..., x_n] = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j>i}^{n}(x_i - x_j)^2,$$

and by cancellation we see that showing $\textbf{Var}[D - x_1] \leq \textbf{Var}[D - x_i]$ is equivalent to,

$$\frac{1}{n^2}\sum_{j\neq 1}(x_i - x_j)^2 \leq \frac{1}{n^2}\sum_{j\neq i}(x_1 - x_j)^2,$$

which is also equivalent to showing,

$$\sum_{j\neq 1,i}(x_i - x_j)^2 \leq \sum_{j\neq 1,i}(x_1 - x_j)^2.$$

The proof then follows from the fact that this is a sum of least squares minimization for the vector $x_2, ..., x_{i-1}, x_{i+1}, ..., x_n$, where we know that $x_1 \leq x_i$ and $\mu(D - x_1 - x_i) \geq \mu(D)$ because $x_1, x_i \leq \mu(D)$. □

We now prove Fact 4.6.3 using the simple helper fact that if we add a data point and want to minimize the variance, then the added data point should the mean of the remaining points.

Fact 4.10.2. *Given any set $x_1, ..., x_n \in \mathbb{R}$ with mean μ, then*

$$\arg\min_y \textbf{Var}[y, x_1, ..., x_n] = \mu.$$

Proof. By definition,

$$\textbf{Var}[x_1, ..., x_n] = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j>i}^{n}(x_i - x_j)^2.$$

The problem we are considering fixes $x_1, ..., x_n$ and minimizes the variable y, so each term in the summation that does not include y can be ignored, and our minimization problem then reduces to,

$$\arg\min_y \textbf{Var}[y, x_1, ..., x_n] = \arg\min_y \sum_{i=1}^{n}(y - x_i)^2,$$

which is minimized when $y = \mu$. $\square$

We use this fact to lower bound the variance from adding one additional variable, and complete our proof of Fact 4.6.3.

Proof of Fact 4.6.3. We first upper bound the variance of all variables:

$$\min_y \textbf{Var}\left[x_1, ..., x_{n-1}, y\right] \leq \textbf{Var}\left[x_1, ..., x_n\right].$$

Fact 4.10.2 implies that,

$$\min_y \textbf{Var}\left[x_1, ..., x_{n-1}, y\right] = \textbf{Var}\left[x_1, ..., x_{n-1}, \mu_{[1:n-1]}\right],$$

where $\mu_{[1:n-1]}$ is the mean of $x_1, ..., x_{n-1}$. We then apply the definition of variance to get,

$$\textbf{Var}\left[x_1, ..., x_{n-1}, \mu_{[1:n-1]}\right] = \frac{1}{n}\sum_{i=1}^{n-1}(x_i - \mu_{[1:n-1]})^2.$$

Together, this implies that,

$$\min_y \textbf{Var}\left[x_1, ..., x_{n-1}, y\right] = \frac{n-1}{n}\textbf{Var}\left[x_1, ..., x_{n-1}\right],$$

which gives our desired result. $\square$

Finally, we also needed the following fact to reduce our running time to $O(n^2)$ for implementation of variance.

Proof of Fact 4.6.7. We utilize the definition of variance as,

$$\textbf{Var}\left[x_1, ..., x_n\right] = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{1}{2}\left(x_i - x_j\right)^2.$$

After cancellation from the scalars we have,

$$\left(\frac{n-1}{n}\right)^2 \textbf{Var}\left[D - x_a\right] + \left(\frac{n-1}{n}\right)^2 \textbf{Var}\left[D - x_b\right] - \left(\frac{n-2}{n}\right)^2 \textbf{Var}\left[D - x_a - x_b\right] + \frac{1}{n^2}(x_a - x_b)^2$$
$$= \frac{1}{n^2}\sum_{i\neq a}\sum_{j\neq a}\frac{1}{2}\left(x_i - x_j\right)^2 + \frac{1}{n^2}\sum_{i\neq b}\sum_{j\neq b}\frac{1}{2}\left(x_i - x_j\right)^2 - \frac{1}{n^2}\sum_{i\neq a,b}\sum_{j\neq a,b}\frac{1}{2}\left(x_i - x_j\right)^2 + \frac{1}{n^2}(x_a - x_b)^2.$$

Separating out within the summation gives,

$$\frac{1}{n^2}\sum_{i\neq a,b}\sum_{j\neq a,b}\frac{1}{2}\left(x_i - x_j\right)^2 + \frac{1}{n^2}\sum_{i\neq a}\left(x_b - x_i\right)^2 + \frac{1}{n^2}\sum_{i\neq b}\left(x_a - x_i\right)^2 + \frac{1}{n^2}(x_a - x_b)^2,$$

which is equivalent to $\textbf{Var}\left[D\right]$ as desired. □

REFERENCES

[1] I. Abraham, D. Durfee, I. Koutis, S. Krinninger, and R. Peng, "On fully dynamic graph sparsifiers," in *Proceedings of the 57th annual Symposium on Foundations of Computer Science, FOCS 2016*, Available at https://arxiv.org/pdf/1604.02094v1.pdf, 2016.

[2] A. Andoni, A. Gupta, and R. Krauthgamer, "Towards (1+ ε)-approximate flow sparsifiers," in *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, Available at http://arxiv.org/abs/1310.3252, SIAM, 2014, pp. 279–293.

[3] D. Durfee, J. Peebles, R. Peng, and A. B. Rao, "Determinant-preserving sparsification of sddm matrices with applications to counting and sampling spanning trees," in *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, 2017, pp. 926–937.

[4] S. Janson, "The numbers of spanning trees, hamilton cycles and perfect matchings in a random graph," *Combinatorics, Probability and Computing*, vol. 3, no. 01, pp. 97–126, 1994.

[5] X. Chen, D. Durfee, and A. Orfanou, "On the complexity of nash equilibria in anonymous games," in *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '15, Portland, Oregon, USA: ACM, 2015, pp. 381–390, ISBN: 978-1-4503-3536-2.

[6] C. Daskalakis and C. Papadimitriou, "Computing equilibria in anonymous games," in *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS '07, Washington, DC, USA: IEEE Computer Society, 2007, pp. 83–93, ISBN: 0-7695-3010-9.

[7] C. Daskalakis and C. H. Papadimitriou, "Discretized multinomial distributions and nash equilibria in anonymous games," in *Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS '08, Washington, DC, USA: IEEE Computer Society, 2008, pp. 25–34, ISBN: 978-0-7695-3436-7.

[8] C. Daskalakis, "An efficient ptas for two-strategy anonymous games," in *Proceedings of the 4th International Workshop on Internet and Network Economics*, ser. WINE '08, Shanghai, China: Springer-Verlag, 2008, pp. 186–197, ISBN: 978-3-540-92184-4.

[9] C. Daskalakis and C. H. Papadimitriou, "On oblivious ptas's for nash equilibrium," in *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, ser. STOC '09, Bethesda, MD, USA: ACM, 2009, pp. 75–84, ISBN: 978-1-60558-506-2.

[10] R. Cummings and D. Durfee, "Individual sensitivity preprocessing for data privacy," vol. abs/1804.08645, 2018.

[11] K. Nissim, S. Raskhodnikova, and A. Smith, "Smooth sensitivity and sampling in private data analysis," in *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, ser. STOC '07, 2007, pp. 75–84.

[12] C. Dwork and J. Lei, "Differential privacy and robust statistics," in *Proceedings of the 41st ACM Symposium on Theory of Computing*, ser. STOC '09, 2009.

[13] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith, "Analyzing graphs with node differential privacy," in *Theory of Cryptography*, ser. TCC '13, 2013, pp. 457–476.

[14] J. Blocki, A. Blum, A. Datta, and O. Sheffet, "Differentially private data analysis of social networks via restricted sensitivity," in *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ser. ITCS '13, 2013, pp. 87–96.

[15] S. Raskhodnikova and A. D. Smith, "Efficient lipschitz extensions for high-dimensional graph statistics and node private degree distributions," in *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS '16, 2016, pp. 495–504.

[16] D. A. Spielman and S.-H. Teng, "A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning," *SIAM Journal on Computing*, vol. 42, no. 1, pp. 1–26, 2013.

[17] C. Borgs, M. Brautbar, J. Chayes, and S.-H. Teng, "A sublinear time algorithm for PageRank computations," in *Algorithms and Models for the Web Graph*, Springer, 2012, pp. 41–53.

[18] R. Andersen, F. Chung, and K. Lang, "Local graph partitioning using PageRank vectors," in *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS '06, Washington, DC, USA: IEEE Computer Society, 2006, pp. 475–486, ISBN: 0-7695-2720-5.

[19] R. Andersen and Y. Peres, "Finding sparse cuts locally using evolving sets," in *Proceedings of the 41st annual ACM symposium on Theory of computing*, ser. STOC '09, Bethesda, MD, USA: ACM, 2009, pp. 235–244, ISBN: 978-1-60558-506-2.

[20] L. Orecchia and N. K. Vishnoi, "Towards an SDP-based approach to spectral methods: A nearly-linear-time algorithm for graph partitioning and decomposition," in *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '11, San Francisco, California: SIAM, 2011, pp. 532–545.

[21] S. O. Gharan and L. Trevisan, "Approximating the expansion profile and almost optimal local graph clustering," in *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, IEEE, 2012, pp. 187–196.

[22] M. R. Henzinger and V. King, "Randomized fully dynamic graph algorithms with polylogarithmic time per operation," *Journal of the ACM*, vol. 46, no. 4, pp. 502–516, 1999, Announced at STOC'95.

[23] J. Holm, K. Lichtenberg, and M. Thorup, "Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity," *Journal of the ACM*, vol. 48, no. 4, pp. 723–760, 2001, Announced at STOC'98.

[24] B. M. Kapron, V. King, and B. Mountjoy, "Dynamic graph connectivity in poly-logarithmic worst case time," in *Symposium on Discrete Algorithms (SODA)*, 2013, pp. 1131–1142.

[25] K. Onak and R. Rubinfeld, "Maintaining a large matching and a small vertex cover," in *Symposium on Theory of Computing (STOC)*, 2010, pp. 457–464.

[26] O. Neiman and S. Solomon, "Simple deterministic algorithms for fully dynamic maximal matching," in *Symposium on Theory of Computing (STOC)*, 2013, pp. 745–754.

[27] S. Baswana, M. Gupta, and S. Sen, "Fully dynamic maximal matching in $O(\log n$ update time," *SIAM Journal on Computing*, vol. 44, no. 1, pp. 88–113, 2015, Announced at FOCS'11.

[28] S. Bhattacharya, M. Henzinger, and G. F. Italiano, "Deterministic fully dynamic data structures for vertex cover and matching," in *Symposium on Discrete Algorithms (SODA)*, 2015, pp. 785–804.

[29] S. Baswana, S. Khurana, and S. Sarkar, "Fully dynamic randomized algorithms for graph spanners," *ACM Transactions on Algorithms*, vol. 8, no. 4, 35:1–35:51, 2012, Announced at ESA'06 and SODA'08.

[30] M. Patrascu, "Towards polynomial lower bounds for dynamic problems," in *Symposium on Theory of Computing (STOC)*, 2010, pp. 603–610.

[31] A. Abboud and V. Vassilevska Williams, "Popular conjectures imply strong lower bounds for dynamic problems," in *Symposium on Foundations of Computer Science (FOCS)*, 2014, pp. 434–443.

[32] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak, "Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture," in *Symposium on Theory of Computing (STOC)*, 2015, pp. 21–30.

[33] A. A. Benczúr and D. R. Karger, "Randomized approximation schemes for cuts and flows in capacitated graphs," *SIAM Journal on Computing*, vol. 44, no. 2, pp. 290–319, 2015.

[34] D. Spielman and S. Teng, "Spectral sparsification of graphs," *SIAM Journal on Computing*, vol. 40, no. 4, pp. 981–1025, 2011, Announced at STOC'04. Available at http://arxiv.org/abs/0808.4134.

[35] J. Batson, D. A. Spielman, N. Srivastava, and S.-H. Teng, "Spectral sparsification of graphs: Theory and algorithms," *Communications of the ACM*, vol. 56, no. 8, pp. 87–94, Aug. 2013.

[36] A. Madry, "Fast approximation algorithms for cut-based problems in undirected graphs," in *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, Available at http://arxiv.org/abs/1008.1975, IEEE, 2010, pp. 245–254.

[37] J. Sherman, "Nearly maximum flows in nearly linear time," in *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, Available at http://arxiv.org/abs/1304.2077, 2013, pp. 263–269.

[38] J. A. Kelner, Y. T. Lee, L. Orecchia, and A. Sidford, "An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations," in *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, Available at http://arxiv.org/abs/1304.2338, 2014, pp. 217–226.

[39] R. Peng and D. A. Spielman, "An efficient parallel solver for SDD linear systems," in *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, ser. STOC '14, Available at http://arxiv.org/abs/1311.3286, New York, New York: ACM, 2014, pp. 333–342, ISBN: 978-1-4503-2710-7.

[40] D. A. Spielman and S.-H. Teng, "Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems," *SIAM Journal on Matrix Analysis and Applications*, vol. 35, no. 3, pp. 835–885, 2014, Available at http://arxiv.org/abs/cs/0607105.

[41] R. Kyng, Y. T. Lee, R. Peng, S. Sachdeva, and D. A. Spielman, "Sparsified cholesky and multigrid solvers for connection laplacians," in *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, ACM, 2016, pp. 842–850.

[42] R. Peng, "Approximate undirected maximum flows in $O(m \operatorname{polylog} n)$ time," in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, Available at http://arxiv.org/abs/1411.7631, 2016, pp. 1862–1867.

[43] I. Koutis, "Simple parallel and distributed algorithms for spectral graph sparsification," in *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2014, pp. 61–66.

[44] J. Sherman, "Breaking the multicommodity flow barrier for $O(\sqrt{\log n})$-approximations to sparsest cut," in *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS '09, Washington, DC, USA: IEEE Computer Society, 2009, pp. 363–372, ISBN: 978-0-7695-3850-1.

[45] L. Orecchia, S. Sachdeva, and N. K. Vishnoi, "Approximating the exponential, the Lanczos method and an $\tilde{O}(m)$-time spectral algorithm for balanced separator," in *Proceedings of the 44th symposium on Theory of Computing*, ser. STOC '12, New York, New York, USA: ACM, 2012, pp. 1141–1160, ISBN: 978-1-4503-1245-5.

[46] R. Kyng, A. Rao, S. Sachdeva, and D. A. Spielman, "Algorithms for Lipschitz learning on graphs," in *Proceedings of The 28th Conference on Learning Theory*, 2015, pp. 1190–1223.

[47] D. Cheng, Y. Cheng, Y. Liu, R. Peng, and S. Teng, "Efficient sampling for Gaussian graphical models via spectral sparsification," *Proceedings of The 28th Conference on Learning Theory*, pp. 364–390, 2015.

[48] J. A. Kelner and A. Levin, "Spectral sparsification in the semi-streaming setting," *Theory of Computing Systems*, vol. 53, no. 2, pp. 243–262, 2013, Announced at STACS'11.

[49] I. Koutis, A. Levin, and R. Peng, "Improved spectral sparsification and numerical algorithms for SDD matrices," in *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, C. Dürr and T. Wilke, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 14, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012, pp. 266–277, ISBN: 978-3-939897-35-4.

[50] M. Kapralov, Y. T. Lee, C. Musco, C. Musco, and A. Sidford, "Single pass spectral sparsification in dynamic streams," in *55th IEEE Annual Symposium on Foundations*

of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014, available at: http://arxiv.org/abs/1407.1289, 2014, pp. 561–570.

[51] Z. A. Zhu, Z. Liao, and L. Orecchia, "Spectral sparsification and regret minimization beyond matrix multiplicative updates," in *Symposium on Theory of Computing (STOC)*, 2015, pp. 237–245.

[52] Y. T. Lee and H. Sun, "Constructing linear-sized spectral sparsification in almost-linear time," in *Symposium on Foundations of Computer Science (FOCS)*, 2015, pp. 250–269.

[53] G. Jindal and P. Kolev, "Faster spectral sparsification of Laplacian and SDDM matrix polynomials," *CoRR*, vol. abs/1507.07497, 2015.

[54] D. Peleg and S. Solomon, "Dynamic $(1 + \epsilon)$-approximate matchings: A density-sensitive approach," in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, 2016, pp. 712–729.

[55] D. A. Spielman and N. Srivastava, "Graph sparsification by effective resistances," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1913–1926, 2011.

[56] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng, "Lower-stretch spanning trees," *SIAM Journal on Computing*, vol. 38, no. 2, pp. 608–628, 2008, Announced at STOC'05. Available at https://www.cs.bgu.ac.il/˜elkinm/sicomp_final.pdf.

[57] J. A. Tropp, "User-friendly tail bounds for sums of random matrices," *Found. Comput. Math.*, vol. 12, no. 4, pp. 389–434, Aug. 2012.

[58] M. K. de Carli Silva, N. J. A. Harvey, and C. M. Sato, "Sparse sums of positive semidefinite matrices," *ACM Transactions on Algorithms*, vol. 12, no. 1, p. 9, 2016.

[59] A. Zouzias, "A matrix hyperbolic cosine algorithm and applications," in *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2012, pp. 846–858.

[60] K. J. Ahn, S. Guha, and A. McGregor, "Spectral sparsification in dynamic graph streams," in *Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 2013, pp. 1–10.

[61] M. Kapralov and D. P. Woodruff, "Spanners and sparsifiers in dynamic streams," in *Symposium on Principles of Distributed Computing (PODC)*, 2014, pp. 272–281.

[62] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu, "A simple, combinatorial algorithm for solving SDD systems in nearly-linear time," in *Proceedings of the*

45th Annual Symposium on Theory of Computing, ser. STOC '13, Available at http://arxiv.org/abs/1301.6628, Palo Alto, California, USA: ACM, 2013, pp. 911–920, ISBN: 978-1-4503-2029-0.

[63] Y. T. Lee and A. Sidford, "Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems," in *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, ser. FOCS '13, Washington, DC, USA: IEEE Computer Society, 2013, pp. 147–156, ISBN: 978-0-7695-5135-7.

[64] W. S. Fung, R. Hariharan, N. J. Harvey, and D. Panigrahi, "A general framework for graph sparsification," in *Proceedings of the forty-third annual ACM symposium on Theory of computing*, https://arxiv.org/abs/1004.4080, ACM, 2011, pp. 71–80.

[65] K. J. Ahn and S. Guha, "Graph sparsification in the semi-streaming model," in *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2009, pp. 328–338.

[66] K. J. Ahn, S. Guha, and A. McGregor, "Analyzing graph structure via linear measurements," in *Symposium on Discrete Algorithms (SODA)*, 2012, pp. 459–467.

[67] ——, "Graph sketches: Sparsification, spanners, and subgraphs," in *Symposium on Principles of Database Systems (PODS)*, 2012, pp. 5–14.

[68] M. Gupta and R. Peng, "Fully dynamic $(1+\epsilon)$-approximate matchings," in *Symposium on Foundations of Computer Science (FOCS)*, 2013, pp. 548–557.

[69] A. Bernstein and C. Stein, "Faster fully dynamic matchings with small approximation ratios," in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, 2016, pp. 692–711.

[70] M. Thorup, "Fully-dynamic min-cut," *Combinatorica*, vol. 27, no. 1, pp. 91–127, 2007, Announced at STOC'01.

[71] M. Thorup and D. R. Karger, "Dynamic graph algorithms with applications," in *Scandinavian Workshop on Algorithm Theory (SWAT)*, 2000, pp. 1–9.

[72] D. R. Karger, "Minimum cuts in near-linear time," *Journal of the ACM*, vol. 47, no. 1, pp. 46–76, Jan. 2000, Announced at STOC'96.

[73] D. Kogan and R. Krauthgamer, "Sketching cuts in graphs and hypergraphs," in *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ACM, 2015, pp. 367–376.

[74] N. Harvey, *Matrix concentration and sparsification*, Workshop on "Randomized Numerical Linear Algebra (RandNLA): Theory and Practice", 2012.

[75] S. Even and Y. Shiloach, "An on-line edge-deletion problem," *Journal of the ACM*, vol. 28, no. 1, pp. 1–4, 1981.

[76] D. Gibb, B. M. Kapron, V. King, and N. Thorn, "Dynamic graph connectivity with improved worst case update time and sublinear space," *CoRR*, vol. abs/1509.06464, 2015.

[77] M. Gupta and A. Sharma, "An $O(log(n))$ fully dynamic algorithm for maximum matching in a tree," *ArXiv preprint arXiv:0901.2900*, 2009.

[78] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd. The MIT Press, 2009, ISBN: 0262033844, 9780262033848.

[79] J. Dabney, B. C. Dean, and S. T. Hedetniemi, "A linear-time algorithm for broadcast domination in a tree," *Networks*, vol. 53, no. 2, pp. 160–169, 2009, Available at http://people.cs.clemson.edu/˜bcdean/bcast_tree.ps.

[80] G. Kirchhoff, "Uber die auflosung der gliechungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer strome gefuhrt wird," in *Poggendorgs Ann. Phys. Chem.*, 1847, pp. 497–508.

[81] N. Goyal, L. Rademacher, and S. Vempala, "Expanders via random spanning trees," in *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '09, New York, New York: Society for Industrial and Applied Mathematics, 2009, pp. 576–585.

[82] A. Asadpour, M. X. Goemans, A. Madry, S. O. Gharan, and A. Saberi, "An o(log n/ log log n)-approximation algorithm for the asymmetric traveling salesman problem," in *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '10, Austin, Texas: Society for Industrial and Applied Mathematics, 2010, pp. 379–389, ISBN: 978-0-898716-98-6.

[83] C. Boutsidis, P. Drineas, P. Kambadur, and A. Zouzias, "A randomized algorithm for approximating the log determinant of a symmetric positive definite matrix," *CoRR*, vol. abs/1503.00374, 2015, Available at: http://arxiv.org/abs/1503.00374.

[84] T. Hunter, A. E. Alaoui, and A. M. Bayen, "Computing the log-determinant of symmetric, diagonally dominant matrices in near-linear time," *CoRR*, vol. abs/1408.1693, 2014, Available at: http://arxiv.org/abs/1408.1693.

[85] I. Han, D. Malioutov, and J. Shin, "Large-scale log-determinant computation through stochastic chebyshev expansions.," in *ICML*, Available at: https://arxiv.org/abs/1606.00942, 2015, pp. 908–917.

[86] J. Kelner and A. Madry, "Faster generation of random spanning trees," in *Proceedings of the 50th annual Symposium on Foundations of Computer Science, FOCS 2009*, Available at https://arxiv.org/abs/0908.1448, 2009, pp. 13–21.

[87] A. Madry, D. Straszak, and J. Tarnawski, "Fast generation of random spanning trees and the effective resistance metric," in *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, Available at http://arxiv.org/pdf/1501.00267v1.pdf, 2015, pp. 2019–2036.

[88] D. Durfee, R. Kyng, J. Peebles, A. B. Rao, and S. Sachdeva, "Sampling random spanning trees faster than matrix multiplication," *CoRR*, vol. abs/1611.07451, 2016.

[89] M. B. Cohen and R. Peng, "ℓ_p row sampling by Lewis weights," in *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, ser. STOC '15, Available at http://arxiv.org/abs/1412.0588, Portland, Oregon, USA: ACM, 2015, pp. 183–192, ISBN: 978-1-4503-3536-2.

[90] M. B. Cohen, "Nearly tight oblivious subspace embeddings by trace inequalities," in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, 2016, pp. 278–287.

[91] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, "Sparsification—a technique for speeding up dynamic graph algorithms," *J. ACM*, vol. 44, no. 5, pp. 669–696, Sep. 1997.

[92] A. A. Benczúr and D. R. Karger, "Approximating s-t minimum cuts in õ(n2) time," in *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96, Philadelphia, Pennsylvania, USA: ACM, 1996, pp. 47–55, ISBN: 0-89791-785-5.

[93] D. A. Spielman and S.-H. Teng, "Spectral sparsification of graphs," *SIAM J. Comput.*, vol. 40, no. 4, pp. 981–1025, Jul. 2011.

[94] M. B. Cohen, J. A. Kelner, J. Peebles, R. Peng, A. Rao, A. Sidford, and A. Vladu, "Almost-linear-time algorithms for markov chains and new spectral primitives for directed graphs," 2017, Accepted to *STOC 2017*. Preprint available at `https://arxiv.org/abs/1611.00755`.

[95] W. Baur and V. Strassen, "The complexity of partial derivatives," *Theoretical Computer Science*, vol. 22, no. 3, pp. 317 –330, 1983.

[96] I. C. F. Ipsen and D. J. Lee, *Determinant approximations*, 2011. eprint: arXiv:1105.0437.

[97] T. Hunter, A. E. Alaoui, and A. M. Bayen, "Computing the log-determinant of symmetric, diagonally dominant matrices in near-linear time," *CoRR*, vol. abs/1408.1693, 2014.

[98] C. Boutsidis, P. Drineas, P. Kambadur, and A. Zouzias, "A randomized algorithm for approximating the log determinant of a symmetric positive definite matrix," *CoRR*, vol. abs/1503.00374, 2015.

[99] A. Broder, "Generating random spanning trees," in *Proceedings of the 30th annual Symposium on Foundations of Computer Science, FOCS 1989*, 1989, pp. 442–447.

[100] D. Aldous, "The random walk construction of uniform spanning trees and uniform labelled trees," in *SIAM Journal on Discrete Mathematics*, 1990, pp. 450–465.

[101] A. Guenoche, "Random spanning tree," *Journal of Algorithms*, vol. 4, no. 3, pp. 214–220, 1983.

[102] V. G. Kulkarni, "Generating random combinatorial objects," *Journal of Algorithms*, vol. 11, no. 2, pp. 185–207, 1990.

[103] C. J. Colbourn, W. J. Myrvold, and E. Neufeld, "Two algorithms for unranking arborescences," *Journal of Algorithms*, vol. 20, no. 2, pp. 268–281, 1996.

[104] N. J. A. Harvey and K. Xu, "Generating random spanning trees via fast matrix multiplication," in *LATIN 2016: Theoretical Informatics*, vol. 9644, 2016, pp. 522–535.

[105] V. V. Williams, "Multiplying matrices faster than coppersmith-winograd," in *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '12, New York, New York, USA: ACM, 2012, pp. 887–898, ISBN: 978-1-4503-1245-5.

[106] A. Schild, "An almost-linear time algorithm for uniform random spanning tree generation," in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2018, Los Angeles, CA, USA: ACM, 2018, pp. 214–227, ISBN: 978-1-4503-5559-9.

[107] N. K. Vishnoi, "Lx = b laplacian solvers and their algorithmic applications," 2012.

[108] C. J. Colbourn, R. P. Day, and L. D. Nel, "Unranking and ranking spanning trees of a graph," *Journal of Algorithms*, vol. 10, no. 2, pp. 271–286, 1989.

[109] R. Burton and R. Pemantle, "Local characteristics, entropy and limit theorems for spanning trees and domino tilings via transfer-impedances," *The Annals of Probability*, pp. 1329–1371, 1993.

[110] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge university press, 2012.

[111] G. Jindal, P. Kolev, R. Peng, and S. Sawlani, "Density independent algorithms for sparsifying k-step random walks," *CoRR*, vol. abs/1702.06110, 2017.

[112] D. D. Sleator and R. E. Tarjan, "Self-adjusting binary search trees," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 652–686, 1985.

[113] S. Alstrup, J. Holm, K. D. Lichtenberg, and M. Thorup, "Maintaining information in fully dynamic trees with top trees," *Acm Transactions on Algorithms (talg)*, vol. 1, no. 2, pp. 243–264, 2005.

[114] J. Nash, "Equilibrium points in n-person games," *Proceedings of the National Academy of Sciences*, vol. 36, no. 1, pp. 48–49, 1950.

[115] ——, "Non-cooperative games," *Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, 1951.

[116] C. Holt and A. Roth, "The Nash equilibrium: A perspective," *Proceedings of the National Academy of Sciences*, vol. 101, no. 12, pp. 3999–4002, 2004.

[117] T. Abbott, D. Kane, and P. Valiant, "On the complexity of two-player win-lose games," in *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, 2005, pp. 113–122.

[118] X. Chen, X. Deng, and S.-H. Teng, "Sparse games are hard," in *Proceedings of the 2nd Workshop on Internet and Network Economics*, 2006, pp. 262–273.

[119] X. Chen, S.-H. Teng, and P. Valiant, "The approximation complexity of win-lose games," in *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007, pp. 159–168.

[120] C. Daskalakis, P. Goldberg, and C. Papadimitriou, "The complexity of computing a Nash equilibrium," *SIAM Journal on Computing*, vol. 39, no. 1, 2009.

[121] X. Chen, X. Deng, and S.-H. Teng, "Settling the complexity of computing two-player Nash equilibria," *Journal of the ACM*, vol. 56, no. 3, pp. 1–57, 2009.

[122] K. Etessami and M. Yannakakis, "On the complexity of Nash equilibria and other fixed points," *SIAM Journal on Computing*, vol. 39, no. 6, pp. 2531–2597, 2010.

[123] R. Mehta, "Constant rank bimatrix games are PPAD-hard," in *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, 2014, pp. 545–554.

[124] R. Lipton, E. Markakis, and A. Mehta, "Playing large games using simple strategies," in *Proceedings of the 4th ACM Conference on Electronic Commerce*, 2004, pp. 36–41.

[125] I. Bárány, S. Vempala, and A. Vetta, "Nash equilibria in random games," in *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, 2005, pp. 123–131.

[126] S. Kontogiannis, P. Panagopoulou, and P. Spirakis, "Polynomial algorithms for approximating Nash equilibria of bimatrix games," in *Proceedings of the 2nd Workshop on Internet and Network Economics*, 2006, pp. 286–296.

[127] C. Daskalakis, A. Mehta, and C. Papadimitriou, "A note on approximate Nash equilibria," in *Proceedings of the 2nd Workshop on Internet and Network Economics*, 2006, pp. 297–306.

[128] ——, "Progress in approximate Nash equilibria," in *Proceedings of the 8th ACM Conference on Electronic Commerce*, 2007, pp. 355–358.

[129] H. Bosse, J. Byrka, and E. Markakis, "New algorithms for approximate Nash equilibria in bimatrix games," in *Proceedings of the 3rd International Workshop on Internet and Network Economics*, 2007, pp. 17–29.

[130] R. Kannan and T. Theobald, "Games of fixed rank: A hierarchy of bimatrix games," in *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007, pp. 1124–1132.

[131] H. Tsaknakis and P. Spirakis, "An optimization approach for approximate Nash equilibria," in *Proceedings of the 3rd International Workshop on Internet and Network Economics*, 2007, pp. 42–56.

[132] T. Feder, H. Nazerzadeh, and A. Saberi, "Approximating Nash equilibria using small-support strategies," in *Proceedings of the 8th ACM Conference on Electronic Commerce*, 2007, pp. 352–354.

[133] S. Kontogiannis and P. Spirakis, "Efficient algorithms for constant well supported approximate equilibria in bimatrix games," in *Proceedings of the 34th International Colloquium on the Automata, Languages and Programming*, 2007, pp. 595–606.

[134] L. Addario-Berry, N. Olver, and A. Vetta, "A polynomial time algorithm for finding Nash equilibria in planar win-lose games," *Journal of Graph Algorithms and Applications*, vol. 11, no. 1, pp. 309–319, 2007.

[135] H. Tsaknakis and P. Spirakis, "Practical and efficient approximations of Nash equilibria for win-lose games based on graph spectra," in *Proceedings of the 6th International Conference on Internet and Network Economics*, 2010, pp. 378–390.

[136] D. Schmeidler, "Equilibrium points of nonatomic games," *Journal of Statistical Physics*, vol. 7, no. 4, pp. 295–300, 1973.

[137] I. Milchtaich, "Congestion games with player-specific payoff functions," *Games and Economic Behavior*, pp. 111–124, 1996.

[138] M. Blonski, "Anonymous games with binary actions," *Games and Economic Behavior*, pp. 171–180, 1999.

[139] ——, "The women of Cairo: Equilibria in large anonymous games," *Journal of Mathematical Economics*, pp. 254–263, 2005.

[140] E. Kalai, "Partially-specified large games," *Proceedings of the 1st International Workshop on Internet and Network Economics*, pp. 3–13, 2005.

[141] C. Papadimitriou and T. Roughgarden, "Computing correlated equilibria in multiplayer games," *Journal of the ACM*, vol. 55, no. 3, pp. 1–29, 2008.

[142] A. Fabrikant, C. Papadimitriou, and K. Talwar, "The complexity of pure Nash equilibria," in *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, 2004, pp. 604–612.

[143] H. Ackermann, H. Röglin, and B. Vöcking, "On the impact of combinatorial structure on congestion games," *Journal of the ACM*, vol. 55, pp. 1–22, 6 2008.

[144] A. Skopalik and B. Vöcking, "Inapproximability of pure Nash equilibria," in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, 2008, pp. 355–364.

[145] C. Daskalakis and C. Papadimitriou, "Computing equilibria in anonymous games," in *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, 2007, pp. 83–93.

[146] C. Daskalakis, "An efficient PTAS for two-strategy anonymous games," in *Proceedings of the 4th International Workshop on Internet and Network Economics*, 2008, pp. 186–197.

[147] C. Daskalakis and C. Papadimitriou, "Discretized multinomial distributions and Nash equilibria in anonymous games," in *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, 2008, pp. 25–34.

[148] ——, "On oblivious PTAS's for Nash equilibrium," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, 2009, pp. 75–84.

[149] ——, "Approximate Nash equilibria in anonymous games," *Journal of Economic Theory*, 2014.

[150] S. Rashid, "Equilibrium points of nonatomic games: Asymptotic results," *Economics Letters*, vol. 12, pp. 7–10, 1983.

[151] J. Ely and W. Sandholm, "Evolution in Bayesian games I," *Theory of Games and Economic Behavior*, vol. 53, pp. 83–109, 2005.

[152] F. Brandt, F. Fischer, and M. Holzer, "Symmetries and the complexity of pure Nash equilibrium," *Journal of Computer and System Sciences*, vol. 75, no. 3, pp. 163–177, 2009.

[153] R. Rosenthal, "A class of games possessing pure-strategy Nash equilibria," *International Journal of Game Theory*, vol. 2, no. 1, pp. 65–67, 1973.

[154] I. Caragiannis, A. Fanelli, N. Gravin, and A. Skopalik, "Efficient computation of approximate pure Nash equilibria in congestion games," in *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, 2011, pp. 532–541.

[155] ——, "Approximate pure Nash equilibria in weighted congestion games: Existence, efficient computation, and structure," in *Proceedings of the 13th ACM Conference on Electronic Commerce*, 2012.

[156] S. Chien and A. Sinclair, "Convergence to approximate Nash equilibria in congestion games," *Games and Economic Behavior*, vol. 71, no. 2, pp. 315–327, 2011.

[157] Y. Azrieli and E. Shmaya, "Lipschitz games," *Mimeo, Ohio State University*, 2011.

[158] Y. Babichenko, "Best-reply dynamics in large binary-choice anonymous games," *Games and Economic Behavior*, vol. 81, pp. 130–144, 2013.

[159] Y. Cai and C. Daskalakis, "On minmax theorems for multiplayer games," in *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms*, 2011, pp. 217–234.

[160] X. Chen, D. Paparas, and M. Yannakakis, "The complexity of non-monotone markets," in *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, 2013, pp. 181–190.

[161] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proceedings of the 3rd Conference on Theory of Cryptography*, ser. TCC '06, 2006, pp. 265–284.

[162] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 34, pp. 211–407, 2014.

[163] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, ser. SP '08, 2008, pp. 111–125.

[164] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig, "Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays," *PLOS Genetics*, vol. 4, no. 8, pp. 1–9, Aug. 2008.

[165] A. Ghosh and A. Roth, "Selling privacy at auction," *Games and Economic Behavior*, vol. 91, pp. 334–346, 2015, Preliminary Version appeared in the Proceedings of the 12th ACM Conference on Electronic Commerce (EC 2011).

[166] Y. Chen, S. Chong, I. A. Kash, T. Moran, and S. Vadhan, "Truthful mechanisms for agents that value privacy," in *Proceedings of the 14th ACM Conference on Electronic Commerce*, ser. EC '13, 2013, pp. 215–232.

[167] R. Cummings, K. Ligett, A. Roth, Z. S. Wu, and J. Ziani, "Accuracy for sale: Aggregating data with a variance constraint," in *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ser. ITCS '15, 2015, pp. 317–324.

[168] K. Nissim, C. Orlandi, and R. Smorodinsky, "Privacy-aware mechanism design," in *Proceedings of the 13th ACM Conference on Electronic Commerce*, ser. EC '12, ACM, 2012, pp. 774–789.

[169] K. Nissim, R. Smorodinsky, and M. Tennenholtz, "Approximately optimal mechanism design via differential privacy," in *Proceedings of the 2012 Conference on Innovations in Theoretical Computer Science*, ser. ITCS '12, 2012, pp. 203–213.

[170] K. Ligett and A. Roth, "Take it or leave it: Running a survey when privacy comes at a cost," in *Proceedings of the 8th International Conference on Internet and Network Economics*, ser. WINE '12, 2012, pp. 378–391.

[171] L. Fleischer and Y.-H. Lyu, "Approximately optimal auctions for selling privacy when costs are correlated with data," in *Proceedings of the 13th ACM Conference on Electronic Commerce*, ser. EC '12, 2012, pp. 568–585.

[172] A. Ghosh, K. Ligett, A. Roth, and G. Schoenebeck, "Buying private data without verification," in *Proceedings of the Fifteenth ACM Conference on Economics and Computation*, ser. EC '14, 2014, pp. 931–948.

[173] R. Cummings, S. Ioannidis, and K. Ligett, "Truthful linear regression," in *Proceedings of The 28th Conference on Learning Theory*, ser. COLT '15, 2015, pp. 448–483.

[174] B. Waggoner, R. Frongillo, and J. Abernethy, "A market framework for eliciting private data," in *Advances in Neural Information Processing Systems 29*, ser. NIPS '15, 2015, pp. 3492–3500.

[175] R. Cummings, D. M. Pennock, and J. Wortman Vaughan, "The possibilities and limitations of private prediction markets," in *Proceedings of the 17th ACM Conference on Economics and Computation*, ser. EC '16, 2016, pp. 143–160.

[176] Z. Jorgensen, T. Yu, and G. Cormode, "Conservative or liberal? Personalized differential privacy," in *Proceedings of the IEEE 31st International Conference on Data Engineering*, 2015, pp. 1023–1034.

[177] M. Alaggan, S. Gambs, and A.-M. Kermarrec, "Heterogeneous differential privacy," *Journal of Privacy and Confidentiality*, vol. 7, no. 6, pp. 127–158, 2017.

[178] H. Li, L. Xiong, Z. Ji, and X. Jiang, "Partitioning-based mechanisms under personalized differential privacy," in *Advances in Knowledge Discovery and Data Mining*, ser. PAKDD '17, 2017, pp. 615–627.

[179] B. Avent, A. Korolova, D. Zeber, T. Hovden, and B. Livshits, "Blender: Enabling local search with a hybrid differential privacy model," in *26th USENIX Security Symposium*, ser. USENIX Security '17, 2017, pp. 747–764.

[180] A. Blum, K. Ligett, and A. Roth, "A learning theory approach to noninteractive database privacy," in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, STOC*, 2008, pp. 609–618.

[181] R. Cummings, K. Ligett, K. Nissim, A. Roth, and Z. S. Wu, "Adaptive learning with robust generalization guarantees," in *29th Annual Conference on Learning Theory*, ser. COLT '16, 2016, pp. 772–814.

[182] R. Bassily and Y. Freund, "Typicality-based stability and privacy," *CoRR*, vol. abs/1604.03336, 2016.

[183] H. Ebadi, D. Sands, and G. Schneider, "Differential privacy: Now it's getting personal," in *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '15, 2015, pp. 69–81.

[184] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith, "What can we learn privately?" In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS '08, 2008, pp. 531–540.

[185] J. Ullman, "Tight lower bounds for locally differentially private selection," arXiv preprint 1802.02638, 2018.

[186] F. McSherry and K. Talwar, "Mechanism design via differential privacy," in *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, 2007, pp. 94–103.