PRACTICAL METHODS FOR HIGH-DIMENSIONAL DATA PUBLICATION WITH DIFFERENTIAL PRIVACY

A Dissertation Presented

by

RYAN MCKENNA

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2022

Robert and Donna Manning College of
Information and Computer Sciences

PRACTICAL METHODS FOR HIGH-DIMENSIONAL DATA PUBLICATION WITH DIFFERENTIAL PRIVACY

A Dissertation Presented

by

RYAN MCKENNA

ACKNOWLEDGMENTS

My time in the PhD program has been quite challenging, but also immensely rewarding and enjoyable. Looking back and reflecting on the journey there are many people I would like to thank. First I would like to thank my advisors Gerome Miklau and Dan Sheldon. I really appreciate the academic freedom I was granted and the support I was given. I learned and grew a lot about how to approach, conduct, and communicate research under your guidance. Second, I would like to thank my additional committee members, Peter Haas and Adam Smith. Peter, I always enjoy hearing about the projects you are involved with, and I appreciate the engagement and advice you have given to my work over the years during our group meetings. Adam, as an inventor of differential privacy, this thesis truly would not have been possible without your groundbreaking work in 2006. Additionally, thank you both for providing valuable feedback during my proposal defense. Third, I would like to thank additional mentors and collaborators I worked with during the program, including Michael Hay, Ashwin Machanavajjhala, Kunal Talwar, Raj Kumar Maity, Yuchao Tao, Dan Zhang, Brett Mullins, Ios Kotsogiannis, Siddhant Pradhan, Cecilia Ferrando, Joie Wu, Arisa Tajima, Garrett Bernstein, Zhiqi Huang, Miguel Fuentes, George Bissias, and David Pujol. I had a lot of fun working with you all and thinking about tough problems together. Fourth, I would like to thank the DREAM lab and all the members in it. I always enjoy our conversations about life, research, fishing, and anything in between. Fifth, I would like to thank the LGRC A303 group I met in the first year and other friends I met in the program: Soumyabrata Pal, Anna Fariha, Pardis Malekzadeh, Shamya Karumbaiah, Iro Moumoulidou, Anil Saini, and Gopal Sharma. Hanging out with you guys made my first year in grad school so much fun and I have a lot of great memories from that time. Sixth, I would like to thank CSCF and Eileen Hammel, for the technical and administrative support during the program. Seventh, I would like to thank Michela Taufer, Stephen Herbein, Travis Johnston, Dylan Chapp, Will Killian, Sean Gray, and the rest of the members of the GCLab. Working with you guys at UD motivated me to pursue the PhD in the first place, and prepared me well for it. Eighth , I would like to thank my mom, dad, and sister. I always look forward to spending time with you guys in Maine and getting out on the boat or up to camp. Ninth, I would like to thank my hometown friends Richard Judge, Greg Ferland, Grant Kern, Luke Wilcox, and Nestor Taylor. I always have fun catching up with you over holiday breaks and reminiscing about our high school days. Tenth, I would like to thank Steven Van Alstine, Dave Ahlberg, William Rodriguez, and others in the microbiology department for the many fun and hilarious game nights. Finally, I would like to thank my fiancée Mrinalini Ghoshal. Your mathematical insights regarding "1=2" and the ubiquity of Benford's law has had a profound impact on me. In all seriousness, your support has been so important in my life and I am so happy that I was able to go through this journey with you. Your presence has helped me maintain balance during the PhD and your own path through the program has been inspirational.

ABSTRACT

PRACTICAL METHODS FOR HIGH-DIMENSIONAL DATA PUBLICATION WITH DIFFERENTIAL PRIVACY

MAY 2022

RYAN MCKENNA

B.Sc., UNIVERSITY OF DELAWARE

M.Sc., UNIVERSITY OF MASSACHUSETTS, AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Gerome Miklau

In recent years, differential privacy has seen significant growth, and has been widely embraced as the dominant privacy definition by the research community. Much progress has been made on designing theoretically principled and practically sound privacy mechanisms. There have even been some real-world deployments of differential privacy, although it has not yet seen widespread adoption. One challenge is that for some problems, there is a gap between the privacy budget required to have a meaningful privacy guarantee and to retain data utility. A second challenge is that many privacy mechanisms have trouble scaling to high-dimensional data, limiting their applicability to real world data.

In this work, we take significant steps towards addressing these challenges, by designing mechanisms and tools that mitigate this gap and scale effectively to high-dimensional settings. This thesis consists of three high-level contributions. In Chapter 3, we present HDMM, a mechanism for linear query answering under differential privacy that scales effectively to large multi-dimensional domains while providing more utility than a large body of prior work. In Chapter 4, we present PrivatePGM, a general-purpose post-processing tool that can estimate a discrete data distribution from noisy observations, improving the utility and scalability of many existing mechanisms at no cost to privacy. In Chapter 5, we present AIM, a mechanism for differentially private synthetic data generation, that leverages PrivatePGM to scale to high-dimensional settings, while introducing a number of novel components to overcome the utility limitations of prior work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

In an increasingly digital world, nearly every aspect of our online presence is tracked and recorded, ranging from our social network, our location history, our search history, the messages we send, the apps we use, the links we click, the videos and movies we watch, and the amount of time we spend on different web pages. This data clearly has immense value to both businesses and research institutions to personalize products and learn about human behavior, but it also contains sensitive information that some people would prefer not to share. Due to regulatory and ethical constraints, these privacy concerns can prevent access to the data for analysis, which in turn slows down research progress and business insights.

Differential privacy [30] offers an appealing solution to this problem: it allows for the analysis of sensitive data, while providing *formal* guarantees about data privacy. It is a mathematical privacy definition with a quantifiable notion of privacy. Informally, the guarantee to individuals is that the output of a differentially private mechanism will be roughly the same (in distribution) whether or not that individual's data was used. The level of closeness required by the definition is determined by the parameter ϵ, which is simply called the "privacy budget". Lower values of ϵ provide a stronger level of privacy protection, where $\epsilon = 1$ is viewed as an acceptable level of privacy protection, $\epsilon = 0.1$ is considered strong privacy protection, and $\epsilon = 10$ is considered weak privacy protection [31].

For individuals, differential privacy is a compelling guarantee, as it provides assurance that they will not face too much *additional* harm if their data is used in a differentially private computation. For data curators, differential privacy is appealing because it inoculates them from risk stemming from a data release. Differential privacy has seen increased real-world adoption in recent years, being embraced by the U.S. Census [65, 2], Google [33], Apple [90], Microsoft [25], LinkedIn [87], Uber [49], and Meta [76]. While these real-world deployments are certainly encouraging, there are several challenges preventing differential privacy from gaining wider-spread adoption, enumerated and discussed below.

Requires specialization Differential privacy is simply a definition that specifies what it means for a given mechanism to be private, but it does not specify how to design such a mechanism. As a result, in order to use differential privacy in a real world application, experts familiar with the field have to carefully understand the domain and problem setting and develop a mechanism for the task at hand. The research community has developed mechanisms for some of the most commonly encountered problems, and the space is constantly expanding as the community develops new techniques. Nevertheless, identifying the right techniques to use and translating research into application requires specialized expertise in differential privacy to get right.

Poor privacy/utility trade-off The privacy budget used in the real-world deployments listed above varied across applications, with Apple using $\epsilon = 8$, LinkedIn using $\epsilon = 34.9$, and the U.S. Census Bureau using $\epsilon = 19.6$ [23]. These large privacy budgets provide weak privacy protection, but were needed to ensure the released data met utility requirements. This problem can be partially alleviated through the development of new and better mechanisms, that provide more utility for the same level of privacy. Indeed, there is a constant effort from the community to push the privacy/utility frontier forward with new techniques. Despite this research, for some problems the best mechanisms available may not be able to simultaneously provide a level of privacy and utility that is acceptable in real-world applications.

Difficulty scaling to high-dimensional settings In additional to privacy and utility, scalability is an important consideration when applying differential privacy in practice. High-dimensional datasets present unique challenges and as a result many mechanisms are limited to low-dimensional settings. This can be a severe hindrance to real-world tasks which often involve high-dimensional data.

1.1 Problem scope and prior work

In this thesis, we focus on two related problems: *linear query answering* and *synthetic data generation* under differential privacy. There has been substantial work on both of these problems, with a variety of mechanisms available for them. Central to both of these problems is the concept of a query *workload*.

1.1.1 Query workloads

At a high level, a workload is simply a specification of all queries an analyst is interested in, and possibly their relative importance. In this thesis, we focus our attention on the class of *linear queries*, an expressive query class that includes queries that count the number of records in a dataset satisfying a given predicate (e.g., "how many individuals in the dataset have salary $\geq 50K$"). Linear query workloads are expressive enough to encode collections of such queries, and can be used to directly compute a wide variety of useful statistical summaries, including histograms, marginals, range queries, and means. Moreover, these statistics can be used to fit a variety of probabilistic models, including Bernoulli distributions, binomial distributions, Poisson distributions, normal distributions, as well as more complex models like Bayesian networks [110] and Markov random fields [20, 8]. Additionally, the statistics needed to train popular machine learning models for classification like logistic regression [77], linear support vector machines [77], naive bayes [77], decision trees [77], random forests [77], xgboost [21], and lightgbm [51] can be approximated well by linear query workloads. These examples demonstrate the expressive capacity of linear query workloads, and highlight their applicability to a wide variety of data analytics tasks.

1.1.2 Mechanisms for linear query answering

In the linear query answering problem, we are given a workload of linear queries, and our goal is to design a mechanism that answers all queries in the workload as accurately as possible while preserving differential privacy. This is a fundamental problem in the field and its study dates back to the invention of differential privacy itself in the seminal work of Dwork et al. [30], and it is still one of the most well-studied problems in the field [9, 44, 80, 59, 14, 28, 29, 11, 79, 12, 32, 43, 40, 112, 107, 60, 114, 101, 84, 56, 106, 105, 83, 82, 108, 104, 58, 22, 3, 100, 26, 57, 47, 6, 84, 111].

One simple way to privately answer a linear query is to use a noise-addition mechanism, such as the Laplace or Gaussian mechanism. There, we compute the true answer to the query, and add carefully calibrated Laplace or Gaussian noise to this quantity, releasing only the noisy answer. For a *single* linear query this is in some sense the best possible mechanism, and this approach can be easily extended to answer a workload of linear queries by invoking a noise addition mechanism for each query in the workload, and by increasing the noise magnitude with the number of queries in the workload. While it is easy to show that this mechanism satisfies differential privacy, it generally has suboptimal utility because it does not exploit correlation between workload queries, as it simply treats each query independently.

A better approach that has been widely adopted and thoroughly studied is based on the `select-measure-reconstruct` paradigm [112, 107, 60, 114, 101, 84, 56, 106, 105, 83, 82, 108, 104, 58, 22, 3, 100, 26, 57, 47, 60]. Mechanisms in this class work by first **selecting** a new set of queries to measure (different from the workload), then privately **measuring** those queries using a noise addition mechanism, and finally **reconstructing** answers to the workload queries from the noisy measurements. The `select-measure-reconstruct` paradigm generalizes the simple noise-addition mechanism described above, which can be defined by simply selecting the workload queries within this framework. More intelligent query selection and reconstruction procedures can lead to orders-of-magnitude improvement in utility, with no cost to privacy.

This class of mechanisms can be naturally partitioned into two groups: *data-independent mechanisms* and *data-dependent mechanisms*, which we discuss separately in the paragraphs below. Data-independent mechanisms `select` queries based only the workload, while data-dependent mechanisms may `select` queries based on both the workload and the data. Generally, data-independent mechanisms provide unbiased answers to the workload queries while data-dependent mechanisms do not. This is a useful statistical property and it allows us to reason about the error distribution of the mechanism independently of the underlying dataset. However, it can often be a worthwhile trade-off to introduce a small amount of bias for reductions in variance, as data-dependent mechanisms do. Empirical studies have found that the best class of techniques to use depends on the privacy budget and amount of data available, with data-independent mechanisms performing best when these are sufficiently large [46]. Theoretical work also supports these findings [80, 79].

Data-independent mechanisms In 2007, Barak et al. [6] proposed the Fourier mechanism for answering workloads containing low-dimensional marginal queries. This mechanism selects and measures a set of "Fourier basis" queries that have the property that the answer to every marginal query in the workload can be expressed as a linear combination of answers to the selected Fourier queries. These Fourier queries provide a compact and non-redundant encoding of the marginal queries in the workload, and require less noise to answer privately than the workload itself, while allowing the workload queries to be reconstructed without magnifying the noise too much. As a result this mechanism provides lower error than a direct application of the Laplace mechanism, as well as other natural baselines.

In 2010, Hay et al. [47] proposed the Hierarchical mechanism for range query workloads. Like the Fourier mechanism, the Hierarchical mechanism is also based on the `select-measure-reconstruct` paradigm. However, this mechanism instead selects hierarchically-structured interval queries which are better tailored for answering range query workloads. As before, this mechanism offers substantially lower error than a direct application of the Laplace mechanism, since the selected hierarchical queries can be answered with much less noise than the workload queries, and any range query in the workload can be reconstructed via a linear combination of a small number of answers to the measured queries. In concurrent work Xiao et al. proposed the Privelet mechanism, which is based on similar principles and offers comparable error rates to the Hierarchical mechanism [57, 46].

While these mechanisms offer significant improvements over a direct application of the Laplace mechanism, they leave room for improvement. In 2010, Li et al. proposed the Matrix Mechanism to generalize and improve other mechanisms in the `select-measure-reconstruct` paradigm. The Matrix Mechanism utilizes a matrix representation for the workload queries and the selected "strategy" queries. For any workload and strategy matrices, the expected total squared error of the Matrix Mechanism can be computed in closed form in terms of elementary matrix operations. As a result, the strategy that minimizes expected error can be selected by solving a numerical optimization problem. By solving this optimization problem, we obtain a set of queries that minimizes the expected error of the mechanism. This clearly improves existing mechanisms using fixed query strategies in principle, but it can be quite expensive to solve this optimization exactly in real-world settings, and as a result the Matrix Mechanism often fails to run in practice.

Follow-up work carefully studied the optimization problem underlying the Matrix Mechanism and proposed various gradient-based algorithms to approximately solve the underlying optimization problem, including LRM [108], EigenDesign [58], GreedyH [56], and COA [107]. These techniques do not necessarily globally optimize the expected error, although they do produce approximately optimal solutions and scale more favorably than the Matrix Mechanism. However, the scalability of these techniques is still limited by the need to represent the workload queries in matrix form, which can be prohibitively expensive in many real-world settings. For example, one of the U.S. Census workloads we study in Chapter 3 would require 22 TB to represent in matrix form, and it is hopelessly infeasible to run these mechanisms on workloads this large.

The special-purpose mechanisms like Fourier, Hierarchical, Privelet and many others [26, 83, 82] avoid the matrix representation of the workload by imposing restrictions on the workload or workload class. Consequently, these mechanisms scale more favorably than the Matrix Mechanism, although this increased scalability comes at the cost of reduced generality, as these mechanisms are limited in the class of workloads they can support. In short there are a lack of methods that simultaneously provide generality to a wide class of input workloads and scalability to large domains, which are both essential in real-world applications.

Data-dependent mechanisms One of the most notable data-dependent mechanisms is MWEM [41], which is an iterative, greedy mechanism for linear query answering. MWEM iteratively selects a small subset of “hard” queries in the workload, one at at time, which it measures privately via the Laplace mechanism. Specifically, MWEM maintains an estimate of the data distribution, and in each round it identifies a single query in the workload that is poorly approximated under the current estimate of the data distribution. By selecting these worst approximated queries MWEM adaptively learns where the estimated data distribution is under-performing, and it makes corrections to improve accuracy on those queries. When the amount of data or privacy budget is small, MWEM can outperform even the best data-independent mechanisms, because it measures a much smaller set of queries, which require less noise to privatize.

Other notable mechanisms inspired by MWEM include DualQuery [36] and FEM [95]. These mechanisms are based on the same underlying principles as MWEM, but are intended to scale more favorably to high-dimensional settings. While these mechanisms do indeed scale better than MWEM, this typically comes at the cost of decreased utility in low-dimensional settings; i.e., DualQuery and FEM are outperformed by MWEM when MWEM is able to run [36].

1.1.3 Mechanisms for synthetic data generation

In the synthetic data generation problem, our goal is to design a mechanism that produces synthetic data that preserves important statistical properties of the original dataset, while exactly matching the domain of the original dataset. The important statistical properties can be most naturally specified through a query workload, and the quality of the underlying synthetic data is evaluated with respect to the query workload. From this perspective, the synthetic data generation problem is closely related to the linear query answering problem, and indeed any mechanism for synthetic data generation can immediately be used for linear query answering, although the reverse is not true in general. For data analysts, synthetic data is a more convenient modality than noisy workload query answers, since they are likely already familiar with the dataset and domain, and have analytics pipelines in place to work with it. Moreover, synthetic data can be used to answer any downstream query, even those that were not specified in the original workload, including complex non-linear queries.[1] Differentially private synthetic data has been identified as an important open problem facing the privacy community, and has been the basis of multiple competitions sponsored by the National Institute of Standards and Technology (NIST) [98, 70, 74]. Tools for synthetic data generation that preserve privacy and utility are highly sought after, as evidence by the large (and growing) list of startup companies offering such services [24].

Many mechanisms for synthetic data generation also follow the `select-measure-reconstruct` paradigm, although in the last step we `reconstruct` synthetic data rather than workload query answers. With some simple modifications to this step, many of the mechanisms described in Section 1.1.2 can be used to generate differentially private synthetic data. However, these mechanisms rely on a vector representation of the dataset, which prevents them from scaling to high-dimensional settings. Indeed, one of the central challenges to the problem of differentially private synthetic data generation is scalability.

One of the first scalable mechanisms for differentially private synthetic data, PrivBayes [110], works by carefully selecting a set of low-dimensional marginal queries to measure, then uses the noisy marginals to fit the parameters of a Bayesian network that is used to sample synthetic records. The Bayesian network provides a compact representation for the data distribution, allowing this mechanism to scale effectively to high-dimensional settings.

Since the first NIST competition in 2019, there has been renewed interest in this problem, and a number of new mechanisms have been proposed [70, 116, 5, 64, 63, 71, 16]. Like PrivBayes, these other mechanisms also scale by selecting low-dimensional marginal queries, although they differ in their methodology for selecting these marginal queries, and they utilize different representations for the underlying data distribution

[1]While synthetic data can be used to answer essentially any query, it will generally provide the best accuracy for queries in the workload.

as well. Among these mechanisms, FEM [95], RAP [5], and GEM [64] select queries based on the workload, while mechanisms like PrivBayes [110], PrivSyn [116], PrivMRF [16], and MST [70] ignore the workload, and instead try to learn the data distribution holistically. Interestingly, the utility of the workload-aware mechanisms is often less than the best workload-agnostic mechanisms, even when utility is measured by the workload [71]. This is a clear suboptimality in existing work and highlights a need for more research in this space.

1.2 Gaps in the literature and thesis contributions

In this section, we reiterate the gaps in existing work and summarize how we overcome them in this thesis.

1.2.1 Lack of general, scalable, and accurate mechanisms for linear queries

Real-world workloads often involve multi-dimensional data and contain complex, customized queries. For example, the U.S. Census publishes demographic statistics about the U.S. population every ten years in the "Summary File 1". While the queries needed to compute these statistics are all linear, there are 215,852 total queries that are defined over a domain with six attributes. Existing special-purpose methods do not support this type of customized workload well, while general-purpose methods do not scale well to these large multi-dimensional domains. Simple baselines like the Laplace mechanism may be able to handle these complex workloads and large domains, but they usually offer suboptimal accuracy.

In Chapter 3, we present the High-Dimensional Matrix Mechanism (HDMM), a new mechanism for linear query answering that offers generality to a wide class of workloads, scalability to large multi-dimensional domains, and state-of-the-art accuracy in a variety of settings. HDMM can be seen as a practical instantiation of the Matrix Mechanism, and is inspired by the same principles that underlie it. HDMM overcomes the main scalability limitations of the Matrix Mechanism by utilizing novel implicit matrix representations for the query workload, and developing new optimization routines that exploit these representations. As a result of these innovations, HDMM enjoys the scalability of special-purpose mechanisms, while also retaining much of the generality and utility of the Matrix Mechanism. In Section 3.10, we show that HDMM consistently outperforms all existing data-independent mechanism on a variety of single- and multi-dimensional domains and workloads, while scaling much better than other general-purpose mechanisms.

1.2.2 No unified method for the `reconstruct` sub-problem

Often, real-world datasets contain even more than six attributes, and many differentially private mechanisms, including HDMM, are unable to scale to these settings. The primary bottleneck of these mechanisms is in the `reconstruct` step, where they estimate the underlying data distribution from the noisy measurements. These methods represent the data distribution in vector form, which is often intractably large for high-dimensional domains.

Moreover, within the `select-measure-reconstruct` paradigm, many mechanisms propose their own techniques for the `reconstruct` subproblem, although they are often based on similar principles. For example, H2 [47], HB [83], Privelet [100], DataCube [26], the Matrix Mechanism [57], and HDMM [69] all solve an ordinary least squares problem to estimate the underlying data distribution, but utilize different customized algorithms to solve this problem tailored to the measured queries. In contrast, Fourier [6], PriView [84], MWEM [41], and DualQuery [36] propose other procedures which are unique to their mechanisms. In addition to these special-purpose approaches to the `reconstruct` subproblem, several general-purpose solutions have been proposed [55, 60, 109]. These existing general-purpose approaches, as well as many special purpose approaches rely on the vector representation of the data, and hence fail to scale to high-dimensional settings. There are a few notable exceptions of techniques that can scale to high-dimensional settings by avoiding the data vector representation, but they have limited applicability [36, 84, 110].

In Chapter 4, we present PrivatePGM, a general and scalable solution to the `reconstruct` subproblem that can replace existing `reconstruct` methods in virtually any mechanism in the `select-measure-reconstruct` paradigm. PrivatePGM is applicable to both the linear query answering problem and the synthetic data generation problem, and can be used for either task. PrivatePGM is most effective when the noisy measurements only depend on the data through its low-dimensional marginals, as this allows it to avoid the intractable vector representation of the data distribution in favor of a more compact representation as a probabilistic graphical model. The assumption that the noisy measurements only depend on the data through its low-dimensional marginals is natural when working with high-dimensional data, and indeed many mechanisms for both linear query answering and synthetic data generation satisfy this assumption [110, 84, 26, 116, 70, 71, 64, 5].

The scalability of PrivatePGM depends on the noisy measurements, and it is capable of scaling up to arbitrarily large domains if the measurements allow it. In Section 4.6, we scale PrivatePGM up to 1000-dimensional domains, far beyond the domains prior work could handle. We also integrate PrivatePGM into existing mechanisms, and show that it consistently boosts both the scalability and utility of these mechanisms. Additionally, because it provides a general and scalable solution to the `reconstruct` subproblem, PrivatePGM can also be used as a component of future mechanisms in the `select-measure-reconstruct` paradigm, allowing future research to think more carefully about the equally important `select` sub-problem. Indeed, in follow-up work, PrivatePGM has been used in exactly this way by MST [70], PrivMRF [16], and AIM [71], the latter of which is presented in Chapter 5.

1.2.3 Lack of workload-aware synthetic data mechanisms

Synthetic data generation is an important open problem facing the community and there has been significant progress on this problem in recent years, with a variety of mechanisms being proposed [110, 116, 16, 70, 95, 5, 64]. However, a careful survey of the field reveals obvious suboptimalities in existing work [71]. In real-world deployments of differential privacy, these suboptimalities hurt utility and lead to an inefficient use of the privacy budget, which is unacceptable in practice, where the privacy budget is considered a precious resource that must be used judiciously.

One common suboptimality of existing mechanisms is workload-awareness. Several mechanisms, including PrivBayes [110], PrivSyn [116], PrivMRF [16], and MST [70], do not consider the workload to be part of a problem statement, and instead attempt generate synthetic data that provides utility on all possible workloads. While this sounds appealing, any mechanism that tries to provide utility for all possible workloads will inevitably have suboptimal utility on the workload that actually matters. Recently, workload-aware synthetic data generation mechanisms have been proposed, including DualQuery [36], FEM [95], RAP [5], and GEM [64]. However, despite the clear drawback of workload-agnostic mechanisms, our experiments in Section 5.6 reveal that they consistently outperform workload-aware mechanisms, even when utility is measured by the workload, a surprising result that highlights a need for more work in this space.

In Chapter 5, we present AIM [71], a new mechanism for workload-aware synthetic data generation, designed by carefully surveying the field and identifying the best elements of existing mechanisms, while also identifying and overcoming the limitations of prior approaches. As a result of these carefully thought out design decisions, AIM consistently outperforms all prior work by significant margins in a wide variety of experimental settings, as we show in Section 5.6. These improvements over prior work are welcomed and needed in real-world deployments of differential privacy in order to achieve better privacy/utility trade-offs.

1.3 Impact

Some of the work presented in this thesis has had real-world impact. For example, HDMM (Chapter 3) was licensed by Tumult Labs, a startup company offering differential privacy consulting and contracting services, and was considered for use by the U.S. Census Bureau in the 2020 decennial census. My expertise on differentially private linear query answering has been recognized by the broader community in various ways. In 2019, I gave a tutorial on linear query answering alongside two other experts in the field, Gerome Miklau and Sasho Nikolov, at the "Data Privacy: Foundations and Applications" seminar at the Simons institute (UC Berkeley). In addition, I contributed a blog post on linear query answering to the NIST Differential Privacy Blog Series [67] in order to make some of the key ideas behind HDMM accessible to a broad audience.

In 2019, I participated in the NIST differential privacy synthetic data competition and developed a new mechanism using PrivatePGM that won first place in this competition [70]. In 2021, I competed in the follow-up temporal map competition on team minutemen, and we finished in second place with another new mechanism built on top of PrivatePGM [74]. Although our team did not win the follow-up competition, the winning team (N-CRiPT) also used PrivatePGM in their solution [16], which is perhaps even better evidence of the impact that PrivatePGM is having on the field. Additionally, I contributed a blog post to `differentialprivacy.org` to provide a high level accessible overview of PrivatePGM and other general-purpose and scalable approaches to the `reconstruct` problem [68], and to encourage synthetic data enthusiasts to use our open source repository. Our open source repository was designed not just to allow researchers to replicate the experiments in our paper, but to easily integrate PrivatePGM (as well as other general-purpose approaches for the `reconstruct` problem) into their own work. It is thoroughly documented, includes a variety of examples and unit tests, and requires minimal dependencies to install.

This effort appears to have been successful, as the repository has been forked and starred dozens of times. My expertise on differentially private synthetic data has been recognized by the community, and I have been invited to give talks on the topic at a variety of venues including the Joint Statistical Meetings, the NIST PSCR Differential Privacy Workshop, the Google Differential Privacy Seminar Series, and several others.

1.4 Additional published work

This thesis draws on work from three papers [69, 73, 71]. Below is a list of additional published research that is not presented in this thesis.

1. **R. McKenna**, S. Pradhan, D. Sheldon, G. Miklau, "Relaxed Marginal Consistency for Differentially Private Query Answering" in Proceedings of 35th International Conference on Neural Information Processing Systems (NeurIPS), 2021.

2. **R. McKenna**, G. Miklau, D. Sheldon, "Winning the NIST Contest: A scalable and general approach to differentially private synthetic data" in Proceedings of the Journal of Privacy and Confidentiality (JPC) special issue on data challenges, 2021.

3. **R. McKenna**, D. Sheldon, "Permute-and-Flip: A new mechanism for differentially private selection" in Proceedings of 34th International Conference on Neural Information Processing Systems (NeurIPS), 2020.

4. **R. McKenna**, R.K. Maity, A. Mazumdar, G. Miklau, "A workload-adaptive mechanism for linear queries under local differential privacy" in Proceedings of 46th International Conference on Very Large Data Bases (VLDB), 2020.

5. D. Pujol, **R. McKenna**, S. Kuppam, M. Hay, A. Machanavajjhala, G. Miklau, "Fair Decision Making using Privacy-Protected Data" in Proceedings of 3rd International Conference on Fairness, Accountability, and Transparency (FAT*), 2020.

6. D. Zhang, **R. McKenna**, I. Kotsogiannis, M. Hay, A. Machanavajjhala, G. Miklau, "Ektelo: A Framework for Defining Differentially-Private Computations," in Proceedings of Special Interest Group on Management of Data (SIGMOD), 2018.

7. G. Bernstein, **R. McKenna**, T. Sun, M. Hay, G. Miklau, D. Sheldon, "Differentially Private Learning of Undirected Graphical Models using CGMs," in Proceedings of 34th International Conference on Machine Learning (ICML), 2017.

CHAPTER 2
BACKGROUND

In this chapter, we introduce the requisite background and notation needed to understand the remainder of this thesis. Below, we provide definitions and background on discrete data, linear queries, and differential privacy.

2.1 Discrete data

A dataset D is a multiset of N records, each containing potentially sensitive information about one individual. Each record $x \in D$ is a d-tuple $(x_1, \dots, x_d)$. The domain of possible values for x_i is denoted by Ω_i, which we assume is finite and has size $|\Omega_i| = n_i$. The full domain of possible values for x is thus $\Omega = \Omega_1 \times \cdots \times \Omega_d$ which has size $\prod_i n_i = n$. We use $\mathcal{D}$ to denote the set of all possible datasets, which is simply $\cup_{N=0}^{\infty} \Omega^N$. It is often convenient to work with the *data vector* representation of D, which we define below.

Definition 1 (Data vector)**.** *The data vector representation of D, denoted $\boldsymbol{p}$, is a vector indexed by tuples $t \in \Omega$, such that $\boldsymbol{p}(t) = \sum_{x \in D} \mathbb{I}[x = t]$.*

Informally, $\boldsymbol{p}(t)$ counts the number of occurrences of t in D. It is an *unnormalized* probability distribution. In high-dimensional domains, representing the data vector in this form is infeasible, as the size of $\boldsymbol{p}$ grows exponentially with dimensionality. In those settings, it is common to instead work with different *marginals* of the data. A marginal for a set of attributes r is essentially a data vector of a low-dimensional projection of D. That is, it is a table that counts the number of occurrences of each $t \in \Omega_r$.

Definition 2 (Marginal vector). *Let $r \subseteq [d]$, $\Omega_r = \prod_{i \in r} \Omega_i$, $n_r = |\Omega_r|$, and $x_r = (x_i)_{i \in r}$. The marginal on r is a vector $\boldsymbol{\mu}_r \in \mathbb{R}^{n_r}$, indexed by domain elements $t \in \Omega_r$, such that each entry is a count, i.e., $\boldsymbol{\mu}_r(t) = \sum_{x \in D} \mathbb{1}[x_r = t]$. We let $M_r : \mathcal{D} \rightarrow \mathbb{R}^{n_r}$ denote the function that computes the marginal on r, i.e., $\boldsymbol{\mu}_r = M_r(D)$.*

Note that the marginal on $r = [d]$ is simply the data vector $\boldsymbol{p}$. In this thesis, we use the term *clique* to refer to the attribute subset r, *marginal query* to denote the function M_r, and *marginal* to denote the vector of counts $\boldsymbol{\mu}_r = M_r(D)$.

2.2 Linear queries

In this thesis, we work extensively with the class of *linear queries.*

Definition 3 (Linear query). *A linear query $q_\phi : \mathcal{D} \rightarrow \mathbb{R}$ is a function that can be expressed as:*

$$q_\phi(D) = \sum_{x \in D} \phi(x),$$

where $\phi : \Omega \rightarrow \mathbb{R}$.

When ϕ is an indicator function, e.g., $\phi(x) = \mathbb{1}[x_{Income} \geq 50K]$, q_ϕ is a predicate counting query. Predicate counting queries are a special-but-common case of linear queries that demonstrate the expressiveness of the query class. A *workload* is simply a collection of m such linear queries, or equivalently a function $W : \mathcal{D} \rightarrow \mathbb{R}^m$. Linear query workloads can express queries to compute histograms, range queries, empirical CDFs, marginals, averages, and many more statistical summaries. Linear queries have a natural vector representation, which can often be more convenient to work with.

Proposition 1 (Query vector). *The linear query q_ϕ can be expressed as a vector $\boldsymbol{q} \in \mathbb{R}^n$, indexed by domain elements $x \in \Omega$, where $\boldsymbol{q}(x) = \phi(x)$. The answer to the linear query can be evaluated as $q_\phi(D) = \boldsymbol{q}^\top \boldsymbol{p}$, where $\boldsymbol{p}$ is the data vector of D. Moreover, if ϕ only depends on x through x_r for some clique $r \subseteq [d]$, then $\phi(x) = \psi(x_r)$*

and the linear query can be expressed as a vector $\boldsymbol{q} \in \mathbb{R}^{n_r}$*, indexed by domain elements* $t \in \Omega_r$*, where* $\boldsymbol{q}(t) = \psi(t)$*. The answer can be evaluated as* $q_\phi(D) = \boldsymbol{q}^\top \boldsymbol{\mu}_r$*, where* $\boldsymbol{\mu}_r = M_r(D)$*.*

Since linear queries can be naturally represented as vectors, linear query workloads can also be naturally represented as matrices, where each row is a query vector. We will use $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ to denote a workload matrix, and can evaluate the query answers by computing the matrix-vector product $W(D) = \boldsymbol{W}\boldsymbol{p}$. Similarly, if W only depends on D through some of it's attributes, we can instead compute $W(D) = \boldsymbol{W}\boldsymbol{\mu}_r = \boldsymbol{W} M_r(D)$.

2.3 Differential privacy

Differential privacy is mathematical privacy definition that requires the output distribution of a randomized algorithm to not differ too much between a dataset D and a neighboring dataset $D' \sim D$. Two datasets $D, D' \in \mathcal{D}$ are neighbors if D' can be obtained from D by adding or removing a single record.

Definition 4 (Differential Privacy [30])**.** *A randomized mechanism* $\mathcal{A}$ *is* (ϵ, δ)*-differentially private if for any dataset* D*, any* $D' \sim D$*, and any subset of possible outputs* $S \subseteq Range(\mathcal{A})$*,*

$$\Pr[\mathcal{A}(D) \in S] \leq \exp(\epsilon) \times \Pr[\mathcal{A}(D') \in S] + \delta.$$

When $\delta = 0$, we say that $\mathcal{A}$ is ϵ-differentially private. One celebrated property of differential privacy is the *post-processing* principle, which says that differentially private mechanisms are robust to post-processing in the sense that post-processing does not affect the privacy guarantee.

Theorem 1 (Postprocessing [31])**.** *Let* $\mathcal{A}$ *be an* (ϵ, δ)*-differentially private mechanism, and let* f *be an arbitrary function, then the mechanism* $\mathcal{A}' = f \circ \mathcal{A}$ *is also* (ϵ, δ)*-differentially private.*

A key quantity needed to reason about the privacy of common randomized mechanisms is the *sensitivity*, defined below.

Definition 5 (Sensitivity)**.** *Let $f : \mathcal{D} \to \mathbb{R}^k$ be a vector-valued function of the input data. The L_p sensitivity of f is $\Delta_p(f) = \max_{D \sim D'} \|f(D) - f(D')\|_p$.*

Note that when f is scalar-valued, the L_p sensitivity is the same for all p. It is common to use $p \in \{1, 2\}$, with the L_1 sensitivity being more common when targeting ϵ-DP and the L_2 sensitivity being more common when targeting (ϵ, δ)-DP. It is easy to verify that both the L_1 and L_2 sensitivity of any marginal query M_r are equal to 1, regardless of the attributes in r. This is because one individual can only contribute a count of one to a single cell of the output vector. The proposition below demonstrates that the sensitivity of any linear query matrix $\boldsymbol{Q}$ can be readily computed as the maximum norm of the columns of $\boldsymbol{Q}$.

Proposition 2 (Sensitivity of a Linear Query Matrix [60])**.** *Let $r \subseteq [d]$ be a clique, $\boldsymbol{Q} \in \mathbb{R}^{m \times n_r}$ be a linear query matrix, and $f_{\boldsymbol{Q}}(D) = \boldsymbol{Q} M_r(D)$. The sensitivity of $f_{\boldsymbol{Q}}$ is:*

$$\Delta_p(f_{\boldsymbol{Q}}) = \max_{x \in \Omega_r} \left[\sum_{z=1}^{m} \boldsymbol{Q}(z, x)^p \right]^{1/p},$$

where $\boldsymbol{Q}(z, x)$ is the entry of $\boldsymbol{Q}$ located at row z and column x.

For reasons that will be clear below, we use $\|\boldsymbol{Q}\|_{\mathcal{L}}$ and $\|\boldsymbol{Q}\|_{\mathcal{G}}$ to denote the L_1 and L_2 sensitivity of $f_{\boldsymbol{Q}}$, respectively. We can use the Laplace mechanism to approximate $f(D)$ in a differentially private manner. We achieve this by adding carefully calibrated Laplace noise to the true answer, where the magnitude of the noise is determined by the privacy parameter ϵ and the L_1 sensitivity $\Delta_1(f)$.

Definition 6 (Laplace Mechanism)**.** *Let $f : \mathcal{D} \rightarrow \mathbb{R}^k$ be a vector-valued function of the input data. The Laplace Mechanism $\mathcal{L}$ adds zero-centered i.i.d. Laplace noise with scale $b\Delta_1(f)$ to each entry of $f(D)$. That is,*

$$\mathcal{L}_f(D) = f(D) + \mathit{Lap}(b \cdot \Delta_1(f))^k.$$

We can also use the Gaussian mechanism to answer $f(D)$ in a differentially private manner, by adding carefully calibrated Gaussian noise instead of Laplace noise. The magnitude of noise is calibrated to the L_2 sensitivity $\Delta_2(f)$, which can be much lower than $\Delta_1(f)$ in some cases.

Definition 7 (Gaussian Mechanism)**.** *Let $f : \mathcal{D} \rightarrow \mathbb{R}^k$ be a vector-valued function of the input data. The Gaussian Mechanism $\mathcal{G}$ adds zero-centered i.i.d. Gaussian noise with scale $\sigma\Delta_2(f)$ to each entry of $f(D)$. That is,*

$$\mathcal{G}_f(D) = f(D) + \mathit{Gaus}(\sigma \cdot \Delta_2(f))^k.$$

While the Laplace and Gaussian mechanism can be used to privately answer a real- or vector-valued function, the Exponential mechanism is another useful primitive that can be used to select a candidate from a finite set that maximizes some objective function with bounded sensitivity.

Definition 8 (Exponential Mechanism)**.** *Let $q_r : \mathcal{D} \rightarrow \mathbb{R}$ be quality score function defined for all $r \in \mathcal{R}$ and let $\epsilon \geq 0$ be a real number. Then the exponential mechanism outputs a candidate $r \in \mathcal{R}$ according to the following distribution:*

$$\Pr[\mathcal{A}(D) = r] \propto \exp\Big(\frac{\epsilon}{2\Delta} \cdot q_r(D)\Big),$$

where $\Delta = \max_{r \in \mathcal{R}} \Delta_1(q_r)$.

To reason about the privacy properties of the mechanisms above, it can be useful to work with an intermediate privacy definition, zero-Concentrated Differential Privacy (zCDP). zCDP offers a clean framework for analyzing the composition of the mechanisms above tightly, while allowing for simple conversions to (ϵ, δ)-DP when needed.

Definition 9 (zero-Concentrated Differential Privacy (zCDP) [13]). *A randomized mechanism $\mathcal{A}$ is ρ-zCDP if for any two neighboring datasets D and D', and all $\alpha \in (1, \infty)$, we have:*

$$D_\alpha(\mathcal{A}(D) \,||\, \mathcal{A}(D')) \leq \rho \cdot \alpha,$$

where $D_\alpha(\cdot \,||\, \cdot)$ is the Rényi divergence of order α between two probability distributions.

The three propositions below state the privacy properties of the Laplace, Gaussian, and Exponential mechanisms.

Proposition 3 (Privacy of the Laplace Mechanism [30]). *The Laplace Mechanism satisfies ϵ-DP for $\epsilon = 1/b$, and ρ-ZCDP for $\rho = 1/2b^2$.*

Proposition 4 (Privacy of the Gaussian Mechanism [13]). *The Gaussian Mechanism satisfies ρ-zCDP for $\rho = 1/2\sigma^2$.*

Proposition 5 (Privacy of the Exponential Mechanism [18]). *The Exponential Mechanism satisfies ϵ-DP and ρ-zCDP for $\rho = \epsilon^2/8$.*

Proposition 6 provides a useful guarantee pertaining to multiple adaptive invocations of zCDP mechanisms. While the proposition below covers 2-fold adaptive composition, it can be inductively applied to obtain analogous k-fold guarantees.

Proposition 6 (Adaptive Composition of zCDP Mechanisms [13]). *Let $\mathcal{A}_1 : \mathcal{D} \to \mathcal{R}_1$ be ρ_1-zCDP and $\mathcal{A}_2 : \mathcal{D} \times \mathcal{R}_1 \to \mathcal{R}_2$ be ρ_2-zCDP. Then the mechanism $\mathcal{A} = \mathcal{A}_2(D, \mathcal{A}_1(D))$ is $(\rho_1 + \rho_2)$-zCDP.*

Finally, to obtain (ϵ, δ)-DP guarantees from ρ-zCDP guarantees, we can invoke Proposition 7 below.

Proposition 7 (zCDP to DP [17])**.** *If a mechanism $\mathcal{A}$ satisfies ρ-zCDP, it also satisfies (ϵ, δ)-differential privacy for all $\epsilon \geq 0$ and*

$$\delta = \min_{\alpha > 1} \frac{\exp\big((\alpha - 1)(\alpha\rho - \epsilon)\big)}{\alpha - 1}\Big(1 - \frac{1}{\alpha}\Big)^{\alpha}.$$

CHAPTER 3

HDMM: OPTIMIZING ERROR OF CONJUNCTIVE LINEAR QUERIES UNDER DIFFERENTIAL PRIVACY

3.1 Motivation

In this chapter, we consider the problem of *linear query answering* under differential privacy. That is, our goal is to release answers to a given workload of linear queries while satisfying differential privacy. In particular, we study the Matrix Mechanism [57], a mechanism in the `select-measure-reconstruct` paradigm that generalizes many existing mechanisms for this task [112, 107, 60, 114, 101, 84, 56, 106, 105, 83, 82, 108, 104, 58, 22, 3, 100, 26, 57, 47, 60]. Just like all methods in this paradigm, the Matrix Mechanism consists of three steps: `select` a set of *strategy* queries, `measure` the strategy queries privately with the Laplace or Gaussian mechanism, and finally `reconstruct` workload query answers via post-processing.

The Matrix Mechanism represents the workload and strategy in matrix form, and the data in vector form. With this representation, the `select`, `measure`, and `reconstruct` steps can be completely defined in the language of linear algebra. In addition, there is a simple formula for the expected error of any selected strategy matrix in terms of elementary matrix operations. This enables the Matrix Mechanism to select the optimal strategy (i.e., the one that offers least expected error) by solving a numerical optimization problem.

Using a matrix to represent a workload is appealing because the representation is expressive enough to capture an arbitrary collection of linear queries queries, and it reveals any structure that may exist between the workload queries. However, the size of the workload matrix is equal to the number of queries times the size of the domain, and it is infeasible to represent large workloads defined over multi-dimensional domains as a matrix. Moreover, solving the optimization problem underlying strategy selection is nontrivial and expensive. As a result, the Matrix Mechanism is not very scalable.

Overview of approach and contributions

This chapter describes the High Dimensional Matrix Mechanism (HDMM), which is a practical instantiation of the Matrix Mechanism, capable of scaling to large multi-dimensional domains. HDMM offers the flexibility and workload-adaptivity of the Matrix Mechanism, while offering the scalability of simpler mechanisms. The three items listed below distinguish HDMM from the Matrix Mechanism:

- The Matrix Mechanism represents query workloads *explicitly*, as fully materialized matrices, while HDMM uses a compact *implicit* matrix representation. This permits a lossless representation of a particular class of queries that avoids a representation exponential in the number of attributes. The implicit representation consists of sub-workload matrices (usually one per attribute) which are used as factors of a Kronecker product. Further, we allow the workload to be expressed as unions of such Kronecker terms. This allows us to represent large multi-dimensional workloads efficiently while maintaining the key benefits that the explicit matrix representation offers.

- The numerical optimization problem at the heart of the Matrix Mechanism is practically infeasible, even for a single attribute with a domain of size 10. HDMM introduces four optimization routines for strategy selection: OPT_0, $\mathsf{OPT}_{\otimes}$, OPT_{+}, and $\mathsf{OPT}_{\mathsf{M}}$. OPT_0 is designed for *explicitly* represented workloads, and can scale to domains as large as 8192. $\mathsf{OPT}_{\otimes}$, OPT_{+}, and $\mathsf{OPT}_{\mathsf{M}}$ are three different techniques for optimizing *implicitly* represented workloads (with implicitly represented strategies), and can scale to significantly larger domains. These optimization routines differ in the space of strategies they consider. In all cases, the strategy search space is chosen so that is expressive enough to encode high-quality strategies, while also enabling tractable optimization.

- We also propose efficient algorithms for the `measure` and `reconstruct` steps of HDMM. In the Matrix Mechanism, these steps are implemented by performing matrix operations with the explicit workload and strategy matrices and the data vector. HDMM exploits the implicit representation of the selected strategies to significantly speed up these steps.

As a result of these distinguishing items, HDMM overcomes the main scalability limitations of the Matrix Mechanism, and runs effectively on both low- and high-dimensional workloads. In fact, in our experiments, we find it has higher accuracy than all prior select-measure-reconstruct techniques, even on input workloads for which the prior techniques were specifically designed. It also achieves reasonable runtime and scales more effectively than prior work that performs non-trivial optimization (see Section 3.10 for a detailed scalability evaluation). The main bottleneck of HDMM is representing the data in vector form, which requires space proportional to the domain size; HDMM can scale to domains as large as 10^9.

Organization

This chapter is organized as follows. In Section 3.2 we setup the problem and introduce the Matrix Mechanism. In Section 3.3, we describe OPT_0, an optimization routine that approximately solves the Matrix Mechanism optimization problem for *explicitly* represented workloads. In Section 3.4, we show how many common workloads over high-dimensional domains can be *implicitly* represented in terms of Kronecker products. In Section 3.5, we describe $\mathsf{OPT}_\otimes$ and OPT_+: two optimization routines that can effectively optimize implicitly represented workloads. In Section 3.6, we show how marginal query workloads can be represented implicitly, using an even more compact representation than the one given in Section 3.4. In Section 3.7, we describe $\mathsf{OPT}_{\mathsf{M}}$, an optimization routine that can efficiently optimize implicitly represented workloads with marginal query strategies. In Section 3.9, we describe the remaining steps of HDMM, including how to exploit the implicit representations to efficiently `measure` the strategy queries and `reconstruct` the answers to the workload queries. We perform an experimental evaluation of HDMM in Section 3.10.

3.2 Problem setup

Our goal is to design a differentially-private mechanism $\mathcal{A} : \mathcal{D} \to \mathbb{R}^m$ that answers a given workload W of m linear queries with low expected total squared error (TSE).

Definition 10 (Expected Error [57]). *Given a workload W of m linear queries and a differentially-private mechanism $\mathcal{A}$, the expected total squared error is:*

$$TSE(W, \mathcal{A}) = \max_{D \in \mathcal{D}} \mathbb{E}[\|W(D) - \mathcal{A}(D)\|_2^2]$$

where the expectation is taken over the randomness in the privacy mechanism $\mathcal{A}$.

In this chapter, our focus will be on unbiased mechanisms which produce correct answers in expectation. For this class of mechanisms, the TSE is the same for all datasets D, making it easy to reason about. As TSE is our main focus in this chapter, we will simply use the term “expected error” when referring to it. We can directly apply the Laplace mechanism (Definition 6) or Gaussian mechanism (Definition 7) to privately answer the query workload. As we show below, we can readily reason about the expected error of both these mechanisms.

Proposition 8 (Error of Laplace and Gaussian mechanisms)**.** *Given a workload W, the Laplace and Gaussian mechanisms are unbiased and have the following expected error:*

$$TSE(W, \mathcal{L}) = 2mb^2 \Delta_1(W)^2, \qquad TSE(W, \mathcal{G}) = m\sigma^2 \Delta_2(W)^2,$$

The proof of this statement follows directly from the definition of the Laplace and Gaussian mechanism, by analyzing the variance of the noise added. As evident by Proposition 8, the expected error of these mechanisms depends crucially on the sensitivity of the query workload W, and if this is large then the TSE will also be large. We now introduce a generalization of these mechanisms that often has better expected error.

3.2.1 The Matrix Mechanism

The core idea of the Matrix Mechanism is to apply either the Laplace of Gaussian mechanism $\mathcal{A}$ on a new query *strategy* Q, then use the noisy answers to the queries in $Q : \mathcal{D} \to \mathbb{R}^p$ to estimate answers to the queries in W. The benefits of this approach will become clear when we reason about the expected error. As the name implies, the Matrix Mechanism relies heavily on the matrix representation of the query workload and query strategy (Proposition 1), denoted $\boldsymbol{W}$ and $\boldsymbol{Q}$ respectively.

Definition 11 (Matrix mechanism [57])**.** *Given a workload $W : \mathcal{D} \to \mathbb{R}^m$ and a strategy $Q : \mathcal{D} \to \mathbb{R}^p$, and a privacy mechanism $\mathcal{A}_Q$ that answers Q (either Laplace or Gaussian), the Matrix Mechanism is defined as:*

$$\mathcal{M}_{W,Q,\mathcal{A}}(D) = \boldsymbol{W}\boldsymbol{Q}^+ \mathcal{A}_Q(D),$$

where $\boldsymbol{W}$ and $\boldsymbol{Q}$ are the workload matrix and strategy matrix, and $\boldsymbol{Q}^+$ is the pseudo-inverse of $\boldsymbol{Q}$.

The privacy of the Matrix Mechanism follows from the privacy of $\mathcal{A}$, since that is the only part of the mechanism that has access to the true data. Under some mild conditions stated below, the Matrix Mechanism is unbiased and the error can be expressed analytically as shown below:

Proposition 9 (Error of the Matrix Mechanism [57])**.** *The Matrix Mechanism is unbiased, i.e.,* $\mathbb{E}[\mathcal{M}_{W,Q,\mathcal{A}}(D)] = W(D)$*, and has the following expected error:*

$$\begin{aligned} TSE(W, \mathcal{M}_{W,Q,\mathcal{A}}) &= TSE(Q, \mathcal{A}) \left\| \boldsymbol{W}\boldsymbol{Q}^{+} \right\|_F^2 \\ &\propto \|\boldsymbol{Q}\|_{\mathcal{A}}^2 \left\| \boldsymbol{W}\boldsymbol{Q}^{+} \right\|_F^2 \end{aligned}$$

as long as $\boldsymbol{W}\boldsymbol{Q}^{+}\boldsymbol{Q} = \boldsymbol{W}$ *and* $\mathcal{A}$ *adds i.i.d. noise with mean* 0*. Here,* $\|\boldsymbol{Q}\|_{\mathcal{A}}$ *is the* L_p *sensitivity of the query matrix* $\boldsymbol{Q}$ *(Proposition 2), where* $p = 1$ *if* $\mathcal{A} = \mathcal{L}$ *and* $p = 2$ *if* $\mathcal{A} = \mathcal{G}$*.*

When invoked with $Q = W$, the Matrix Mechanism is very similar to base mechanism $\mathcal{A}$, but the error is typically lower. The term $\|\boldsymbol{W}\boldsymbol{W}^{+}\|_F^2$ in the error formula is equal to the rank of $\boldsymbol{W}$, which is bounded above by m. This implies that the error of the Matrix Mechanism can never be higher than the error of $\mathcal{A}$, when using the workload as the strategy. Further, there are often much better strategies to select than $Q = W$.

Finding the best strategy $\boldsymbol{Q}$ for a given workload $\boldsymbol{W}$ is the main technical challenge of the Matrix Mechanism. This strategy selection problem can be formulated as a constrained optimization problem, and in fact we can immediately map the analytic error formula from Proposition 9 into a numerical optimization problem as follows:

Problem 1 (Matrix Mechanism Optimization [60])**.** *Given an* $m \times n$ *workload matrix* $\boldsymbol{W}$*:*

$$\begin{aligned} \underset{\boldsymbol{Q}}{\text{minimize}} \quad & \|\boldsymbol{Q}\|_{\mathcal{A}}^2 \left\| \boldsymbol{W}\boldsymbol{Q}^{+} \right\|_F^2 \\ \text{subject to} \quad & \boldsymbol{W}\boldsymbol{Q}^{+}\boldsymbol{Q} = \boldsymbol{W} \end{aligned} \tag{3.1}$$

For a number of reasons, this optimization problem is difficult to solve exactly: it has many variables, it is not convex, and both the objective function and constraints involve $\boldsymbol{Q}^+$, which can be slow to compute. In addition, $\|\boldsymbol{W}\boldsymbol{Q}^+\|_F^2$ has points of discontinuity near the boundary of the constraint $\boldsymbol{W}\boldsymbol{Q}^+\boldsymbol{Q} = \boldsymbol{W}$. This problem was originally formulated as a rank-constrained semi-definite program [57], and, while algorithms exist to find the global optimum, they require $O(m^4(m^4+n^4))$ time, making it infeasible in practice.

3.2.2 Lower bounds on error

An important theoretical question is to identify, or bound, how low the error of the Matrix Mechanism can be for a given workload $\boldsymbol{W}$. This is useful because finding the strategy with minimum error is a difficult (and often intractable) problem, but computing a lower bound on error can be done efficiently. Knowing how low the error can be allows one to compare the error of a concrete strategy to the lower bound to see how close to optimal it is. Additionally, the lower bound can be used to make important policy decisions, such as setting the privacy budget, or whether it is worth investing the resources to find a good strategy (as opposed to using other types of mechanisms like data-dependent ones). Li and Miklau studied this problem and derived the SVD bound [59].

Definition 12 (SVD Bound [59])**.** *Given a $m \times n$ workload matrix $\boldsymbol{W}$, the singular value bound is:*

$$SVDB(\boldsymbol{W}) = \frac{1}{n}\big(\lambda_1 + \cdots + \lambda_n\big)^2$$

where $\lambda_1, \ldots, \lambda_n$ are the singular values of $\boldsymbol{W}$.

The SVD bound gives a lower bound on the error achievable by the Matrix Mechanism.

Proposition 10 (SVD Bound [59]). *Given a $m \times n$ workload matrix $\boldsymbol{W}$ and a $p \times n$ strategy matrix $\boldsymbol{Q}$ that supports $\boldsymbol{W}$:*

$$SVDB(\boldsymbol{W}) \leq \|\boldsymbol{Q}\|_{\mathcal{A}}^2 \left\|\boldsymbol{W}\boldsymbol{Q}^+\right\|_F^2$$

The SVD Bound is known to be tight for the Gaussian mechanism (i.e., when $\mathcal{A} = \mathcal{G}$) and some conditions on $\boldsymbol{W}$ are satisfied, meaning there is some strategy $\boldsymbol{Q}$ that achieves the equality. When $\mathcal{A} = \mathcal{L}$ the bound may not be tight however. We will later use the SVD Bound to evaluate the quality of the strategies found by our optimization routines. We also derive expressions for efficiently computing the SVD Bound for implicitly represented workloads, and use this analysis to theoretically justify our optimization routines.

3.3 Optimizing explicit workloads

In this section, we investigate the main optimization problem that underlies the Matrix Mechanism, and describe OPT_0, our algorithm for approximately solving it. We assume for now that the workload is represented *explicitly* as a dense matrix. The methods we describe are useful by themselves for workloads defined over modest domains (namely, those smaller than about $n = 10^4$), and they are an essential building block for the more scalable methods we describe in Section 3.5.

Instead of formulating and solving the intractable SDP as originally proposed, we will attack this problem directly with gradient-based numerical optimization techniques. While these methods do not guarantee convergence to a global optimum of Problem 1, they can be used heuristically to find locally optimal solutions. These techniques begin by guessing a solution $\boldsymbol{Q}_0$ and then iteratively improve it using the gradient of the objective function to guide the search. The process ends after a number of iterations are performed, controlled by a stopping condition based on improvement of the objective function.

In the next sections, we provide algorithms for efficiently solving Problem 1. We separately consider the two cases of Gaussian noise and Laplace noise, as the required techniques are quite different.

3.3.1 Strategy optimization with Gaussian noise

While Problem 1 with $\mathcal{A} = \mathcal{G}$ is not convex in its current form, it can be reformulated into an equivalent problem that is convex [60, 107]. The key idea is that the objective function can be expressed in terms of $\boldsymbol{X} = \boldsymbol{Q}^\top \boldsymbol{Q}$, since $\|\boldsymbol{Q}\|_{\mathcal{G}}^2 = max(diag(\boldsymbol{Q}^\top \boldsymbol{Q}))$ and $\|\boldsymbol{W}\boldsymbol{Q}^+\|_F^2 = tr[(\boldsymbol{Q}^\top \boldsymbol{Q})^+ (\boldsymbol{W}^\top \boldsymbol{W})]$. This allows us to optimize $\boldsymbol{X}$ instead of $\boldsymbol{Q}$, and then we can recover $\boldsymbol{Q}$ by performing Cholesky decomposition on $\boldsymbol{X}$. Remarkably, the resulting problem is convex with respect to $\boldsymbol{X}$.

Definition 13 (Convex Reformulation [107])**.** *Given a workload matrix $\boldsymbol{W}$ of rank n, let* $\mathsf{OPT}_0(\boldsymbol{W}) = \boldsymbol{Q}$ *where $\boldsymbol{Q}^\top \boldsymbol{Q}$ is a Cholesky decomposition of $\boldsymbol{X}^*$ and:*

$$\begin{aligned} \boldsymbol{X}^* = &\underset{\boldsymbol{X}}{minimize} && tr[\boldsymbol{X}^{-1}(\boldsymbol{W}^\top \boldsymbol{W})] \\ &subject\ to && diag(\boldsymbol{X}) = \mathbf{1} \\ & && \boldsymbol{X} \succ 0 \end{aligned} \tag{3.2}$$

While the above problem is convex, it is still nontrivial to solve due to the dependence on the matrix inverse and the constraint $\boldsymbol{X} \succ 0$ ($\boldsymbol{X}$ is positive definite). The equality constraint $diag(\boldsymbol{X}) = 1$ and corresponding optimization variables $diag(\boldsymbol{X})$ can easily be eliminated from the problem since they must always equal 1. Additionally $\boldsymbol{X}$ must be a symmetric matrix so we can optimize over the lower triangular entries, essentially reducing the number of optimization variables by a factor of two. The full details of a conjugate gradient algorithm for solving this problem are available in [107]. The per-iteration runtime of their "COA" algorithm is $O(n^3)$, and by default it runs for 50 iterations. In practice it is able to scale up to about $n \approx 10^4$. The algorithm works well in practice when n is small, but is not particularly robust for some workloads when n is larger, which we observe empirically in Section 3.10 (e.g., COA on prefix workload in Table 3.3).

We thus design our own algorithm to solve the same optimization problem, which is based on the same principles as the COA technique, but is more robust in practice. There are two key differences in our implementation. First, we initialize the optimization intelligently by setting $\boldsymbol{X} = \boldsymbol{P}\sqrt{\boldsymbol{\Lambda}}\boldsymbol{P}^\top$ where $\boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^\top$ is the eigen-decomposition of $\boldsymbol{W}^\top\boldsymbol{W}$. This an approximation to the optimal strategy based on the SVD bound [59], and acts as a very good initialization. Second, instead of using the custom-designed conjugate gradient algorithm proposed in [107], we simply use the L-BFGS algorithm implemented in `scipy.optimize`, an off-the-shelf optimizer. We heuristically ignore the constraint $\boldsymbol{X} \succ 0$ during optimization, using a large loss value when it is not satisfied. This makes the problem unconstrained, and readily solvable by `scipy.optimize`. The constraint is verified to hold at the end of the optimization. These changes lead to more robust optimization that produce strategies nearly matching the SVD bound, as we show experimentally in Section 3.10.

3.3.2 Strategy optimization with Laplace noise

Unfortunately, the techniques used in the previous section do not apply to the Laplace noise setting, as the sensitivity norm $\|\boldsymbol{Q}\|_{\mathcal{L}}$ cannot be expressed in terms of $\boldsymbol{Q}^\top\boldsymbol{Q}$. In this section, we describe an alternate approach to approximately solve Problem 1: parameterized strategies. Our key idea is to judiciously restrict the search space of the optimization problem to simplify the optimization while retaining expressivity of the search space. While our approach does not necessarily produce a globally optimal solution to Problem 1, with good parameterizations it can still find state of the art strategies. Below we describe the idea of parameterized strategies in its full generality. Then we propose a specific parameterization that works well for a variety of input workloads.

A parameterization is a function $\boldsymbol{Q}(\boldsymbol{\theta})$ mapping a real-valued parameter vector $\boldsymbol{\theta}$ to a strategy $\boldsymbol{Q}$. Optimizing over a parameterized strategy space can be performed by optimizing $\boldsymbol{\theta}$ rather than $\boldsymbol{Q}$. In the extreme case, there may be one entry in $\boldsymbol{\theta}$ for every entry in $\boldsymbol{Q}$, but a more careful design of the parameterization with fewer parameters and a smart mapping between the entries of the parameter vector and the entries of the strategy matrix can lead to more effective optimization. There are several design considerations for setting up a good parameterization. First, the parameterization must be expressive enough to encode high-quality strategies (this may depend on the workload). Second, the parameterization should have structure that makes the optimization problem more computationally tractable (such as eliminating constraints). Third, the parameterization may encode domain expertise about what a good strategy should look like, which could make it easier to find high-quality local minima.

We now present a new general-purpose parameterization, called p-Identity, which handles these considerations without making strict assumptions about the structure of the workload. It also out-performs all of the existing parameterizations, even on the workloads for which they were designed. The parameters of a p-Identity strategy are more naturally interpreted as a matrix $\boldsymbol{\Theta}$ rather than a vector $\boldsymbol{\theta}$ so we instead use the notation $\boldsymbol{Q}(\boldsymbol{\Theta})$.

Definition 14 (p-Identity strategies)**.** *Given a $p \times n$ matrix of non-negative values $\boldsymbol{\Theta}$, the p-Identity strategy matrix $\boldsymbol{Q}(\boldsymbol{\Theta})$ is defined as follows:*

$$\boldsymbol{Q}(\boldsymbol{\Theta}) = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{\Theta} \end{bmatrix} \boldsymbol{D}$$

where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{D} = diag(1 + \boldsymbol{1}^\top \boldsymbol{\Theta})^{-1}$.

Intuitively, p-Identity strategies encode $n+p$ queries, including n weighted identity queries that count the number of records in the database for each domain element, as well as p arbitrary linear queries determined by $\boldsymbol{\Theta}$. The diagonal matrix $\boldsymbol{D}$ is used to re-weight the strategy so that $\|\boldsymbol{Q}\|_{\mathcal{L}} = 1$ and each column of $\boldsymbol{Q}$ has the same L_1 norm. This is an example of domain knowledge incorporated into the parameterization, as it has been shown that optimal strategies have uniform column norm [60]. [1]

Example 1. *For $p = 2$ and $n = 3$, we illustrate below how $\boldsymbol{Q}(\boldsymbol{\Theta})$ is related to its parameter matrix, $\boldsymbol{\Theta}$.*

$$\boldsymbol{\Theta} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix} \qquad \boldsymbol{Q}(\boldsymbol{\Theta}) = \begin{bmatrix} 0.33 & 0 & 0 \\ 0 & 0.25 & 0 \\ 0 & 0 & 0.2 \\ 0.33 & 0.5 & 0.6 \\ 0.33 & 0.25 & 0.2 \end{bmatrix}$$

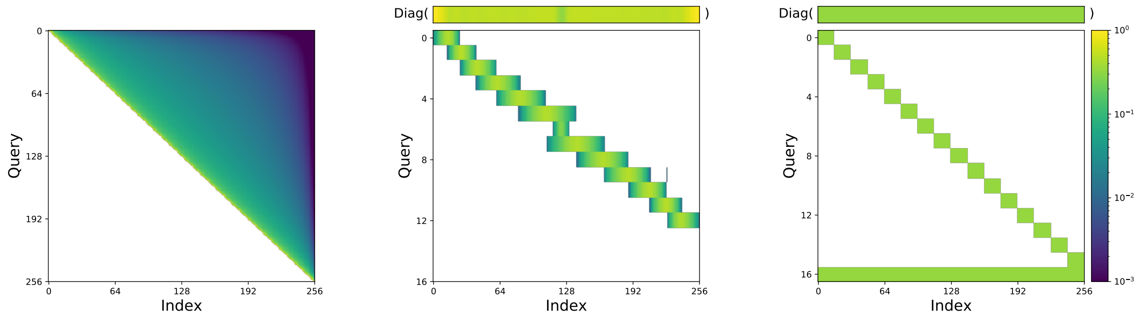
For this class of parameterized strategies, the resulting optimization problem requires optimizing $\boldsymbol{\Theta}$ instead of $\boldsymbol{Q}$ and is stated below; we use OPT_0 to denote the operator that solves this problem.

Definition 15 (parameterized optimization). *Given a workload matrix $\boldsymbol{W}$ and hyper-parameter p, let $\mathsf{OPT}_0(\boldsymbol{W}) = \boldsymbol{Q}(\boldsymbol{\Theta}^*)$ where:*

$$\boldsymbol{\Theta}^* = \underset{\boldsymbol{\Theta} \in \mathbb{R}_+^{p \times n}}{argmin} \quad \left\|\boldsymbol{W}\boldsymbol{Q}(\boldsymbol{\Theta})^+\right\|_F^2$$

This parameterization was carefully designed to simplify optimization. Because $\boldsymbol{Q}(\boldsymbol{\Theta})$ is full rank, the constraints are satisfied by construction, and as a result the only constraint we need to handle is non-negativity of $\boldsymbol{\Theta}$. Furthermore, $\|\boldsymbol{Q}(\boldsymbol{\Theta})\|_1 = 1$ for all $\boldsymbol{\Theta}$, so that term can be removed from the objective. Additionally, due to the special structure of $\boldsymbol{Q}(\boldsymbol{\Theta})$ we can efficiently evaluate the objective and its gradient in $O(pn^2)$ time instead of $O(n^3)$ time. For example, when $n = 8192$ it requires > 6 minutes to evaluate the objective for a general strategy $\boldsymbol{Q}$, while it takes only 1.5 seconds for a p-Identity strategy (with $p = \frac{n}{16}$), which is a $240\times$ improvement. Despite this imposed

[1] If a strategy did not, a query could be added to it without increasing sensitivity and addition of that query would result in error less than or equal to that of the original strategy.



(a) OPT_0 (Gaussian Noise) (b) OPT_0 (Laplace Noise) (c) Hierarchical (b=16)

Figure 3.1: Visualization of three different strategies for answering the workload of all range queries on a domain of size 256. Figures (a) and (b) show strategies optimized by HDMM, and Figure (c) shows a hierarchical strategy with branching factor 16, a previously state-of-the-art strategy for this workload [47, 83]. Each row is a query, and cells are color coded according to their value in the strategy matrix. Figure (a) plots each query as one very thin row, while Figure (b) and (c) only plot the non-trivial queries as thicker rows. The diagonal queries are plotted separately as a single row above the main plot, but it actually represents 256 different queries.

structure, $\boldsymbol{Q}(\boldsymbol{\Theta})$ is still expressive enough to encode high-quality strategies. Moreover, p can always be tuned to balance expressivity with efficiency.

3.3.3 Strategy visualization

An interested reader may wonder what the optimized strategies actually look like for some common workloads. In Figure 3.1, we plot three strategies designed for the workload of all range queries. For an ordered domain $\Omega = \{1, \dots, n\}$, this workload contains $n(n+1)/2$ queries that count the number of records in the interval $[i, j]$ for all $1 \leq i \leq j \leq n$. Figures 3.1a and 3.1b show the strategies produced by HDMM for Gaussian and Laplace noise, respectively, and Figure 3.1c shows a previously state-of-the-art strategy for range queries. These visualizations provide interesting insight into the nature of the solution, and reveal why HDMM succeeds in reducing error.

In Figure 3.1a, the strategy is an upper triangular square matrix; this is by construction as it is obtained through a Cholesky decomposition.[2] In each row, weights are largest near the main diagonal, and quickly decreases the further away it gets. This can be observed from the smooth transition from yellow to green to blue in Figure 3.1a, and noting that the colors appear on a logarithmic scale.

In Figure 3.1b, the strategy was optimized with $p = 16$, but only 13 non-zero queries were found. Each query has varying width, ranging between about 16 and 64, and has greatest weight towards the middle of the query, with gradually decreasing weights away from the center. In most cases, the queries overlap with half of the two neighboring queries. The weights on the identity queries are approximately uniform throughout at around 0.5, with higher weights near the edges. There is a very natural reason why this structure works well for range queries. Summing up adjacent queries leads to a bigger query which looks approximately like a range query. It will have a long uniform center, and decaying weights on the edges. These decaying weights can be increased to match the uniform center by drawing on the answers from the identity queries. Thus, any range query can be answered by summing up a relatively small number of strategy query answers.

This same intuition was used in the derivation of the hierarchical strategy in Figure 3.1c. However, this strategy answers some range queries more effectively than others. It struggles for queries that require summing up many identity queries. For example, it can answer the range $[0, 16)$ using one strategy query, but it requires summing up 16 strategy queries to answer the range $[8, 24)$.

[2]There may be equally good strategies that do not have this upper triangular structure, but HDMM will always find one with this structure.

3.4 Implicit representations for conjunctive linear queries

The optimization methods we described in the previous section work well for small and modest domain sizes; we were able to run them on domains as large as $n = 8192$ (see Section 3.10 for a scalability analysis). However, these methods are fundamentally limited by the need to represent the workload and strategy explicitly in matrix form. It requires 0.5 gigabytes just to store a square matrix of size 8192 using 4 byte floats, and it is time consuming to perform nontrivial matrix operations on objects of this size. This limitation is not unique to our mechanism, but is shared by *all possible* methods that rely on explicitly represented workload matrices.

To overcome this scalability limitation, we propose *implicit query matrices*, which exploit structure in conjunctive linear query sets and offer a far more concise representation than materialized explicit matrices, while still being able to encode query sets containing an arbitrary collection of conjunctive linear queries. These representations are lossless; they save space by avoiding significant redundancy, rather than making approximations. As we will show later in this section, many important matrix operations can be performed efficiently using the implicit representation. This property of the representation will be essential for solving the strategy optimization problem efficiently on large domains.

3.4.1 Implicitly vectorized conjunctive linear queries

We begin by defining a conjunctive linear query, and then show how we will represent it implicitly.

Definition 16 (Conjunctive Linear Query)**.** *A linear query $q_\phi : \mathcal{D} \rightarrow \mathbb{R}$ is said to be conjunctive, iff for all $x \in \Omega$,*

$$\phi(x) = \prod_{i=1}^{d} \psi_i(x_i)$$

The term "conjunctive" above stems from the observation that if ψ_i are all indicator functions, then multiplication and "logical-and" are equivalent. As we demonstrate below, conjunctive linear queries can be naturally represented in vector form in terms of an outer product.

Definition 17 (Outer product)**.** *Let $\boldsymbol{q}_i \in \mathbb{R}^{n_i}$ for $i = 1, \dots, d$. The outer product $\mathbb{q} = \boldsymbol{q}_1 \otimes \cdots \otimes \boldsymbol{q}_d$ is an n-length vector indexed by tuples $x = (x_1, \dots, x_d)$ such that:*

$$\mathbb{q}(x) = \prod_{i=1}^{d} \boldsymbol{q}_i(x_i)$$

The correspondence between conjunctive linear queries and outer products is immediate, as both are defined by the same exact formula. While the explicit vector representation of q_ϕ has size $\prod_i n_i$, the implicit representation only requires storing d smaller query vectors, which in total have size $\sum_i n_i$. We can represent workloads that contain a Cartesian product of conjunctive linear queries even more efficiently using Kronecker products.

Definition 18 (Kronecker product)**.** *Let $\boldsymbol{Q}_i \in \mathbb{R}^{m_i \times n_i}$ for $i = 1, \dots, d$. The Kronecker product, denoted $\mathbb{Q} = \boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d$ is an $m \times n$ matrix where $m = \prod_i m_i$ and $n = \prod_i n_i$, indexed by d-tuples $z = (z_1, \dots, z_d)$ and $x = (x_1, \dots, x_d)$ with entries defined by:*

$$\mathbb{Q}(z, x) = \prod_{i=1}^{d} \boldsymbol{Q}_i(z_i, x_i)$$

The Kronecker product is a generalization of the outer product, and hence we use the same symbol $\otimes$ for both operations. In particular, each row of $\mathbb{Q}$ is an outer product between one row from each of $\boldsymbol{Q}_1, \dots, \boldsymbol{Q}_d$. The table below summarizes the cost of different representational choices. All workloads of conjunctive linear queries can be represented as a list of outer products, and this is far more efficient than an explicit representation. When applicable, the Kronecker product provides an even more compact representation for these workloads.

Explicit Representation	List of Outer Products	Kronecker Product
$(\prod_i m_i)(\prod_i n_i)$	$(\prod_i m_i)(\sum_i n_i)$	$\sum_i m_i n_i$

3.4.2 Operations on vectorized objects

Reducing the size of the workload representation is only useful if critical computations can be performed without expanding them to their explicit representations. Standard properties of the Kronecker product [94] accelerate strategy selection and reconstruction. The following properties allow us to perform useful operations on Kronecker products without materializing their full matrices.

Proposition 11 (Kronecker identities)**.** *Kronecker products satisfy the following identities [94]:*

Transpose:	$(\boldsymbol{A} \otimes \boldsymbol{B})^T = \boldsymbol{A}^T \otimes \boldsymbol{B}^T$
Pseudo Inverse:	$(\boldsymbol{A} \otimes \boldsymbol{B})^+ = \boldsymbol{A}^+ \otimes \boldsymbol{B}^+$
Associativity:	$(\boldsymbol{A} \otimes \boldsymbol{B}) \otimes \boldsymbol{C} = \boldsymbol{A} \otimes (\boldsymbol{B} \otimes \boldsymbol{C})$
Mixed Product:	$(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{C} \otimes \boldsymbol{D}) = (\boldsymbol{AC}) \otimes (\boldsymbol{BD})$

In addition to the standard properties of Kronecker product above, various matrix norms of a Kronecker product can be readily computed from the norms of its factors.

Proposition 12 (Norm of a Kronecker product [54])**.** *The following matrix norms decompose over the factors of the Kronecker product $\boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d$.*

$$\|\boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d\|_{\mathcal{L}} = \prod_{i=1}^{d} \|\boldsymbol{Q}_i\|_{\mathcal{L}}$$
$$\|\boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d\|_{\mathcal{G}} = \prod_{i=1}^{d} \|\boldsymbol{Q}_i\|_{\mathcal{G}}$$
$$\|\boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d\|_{F} = \prod_{i=1}^{d} \|\boldsymbol{Q}_i\|_{F}$$

We will later use these identities to efficiently evaluate TSE for workloads and strategies built with Kronecker products.

3.5 Optimizing conjunctive linear query workloads

We now turn our attention to optimizing implicitly-represented conjunctive linear query workloads. We assume the workload is a *union of Kronecker products*, and that it takes the form shown in Equation (3.3):

$$\mathbb{W} = \begin{bmatrix} w_1 \mathbb{W}_1 \\ \vdots \\ w_k \mathbb{W}_k \end{bmatrix} = \begin{bmatrix} w_1(\boldsymbol{W}_1^{(1)} \otimes & \dots & \otimes \boldsymbol{W}_d^{(1)}) \\ \vdots & \ddots & \vdots \\ w_k(\boldsymbol{W}_1^{(k)} \otimes & \dots & \otimes \boldsymbol{W}_d^{(k)}) \end{bmatrix} \tag{3.3}$$

Workloads of this form are expressive enough to encode an arbitrary collection of conjunctive linear queries. For notational convenience, we will often write $\mathbb{W} = w_1 \mathbb{W}_1 + \cdots + w_k \mathbb{W}_k$, where $\mathbb{W}_i = \boldsymbol{W}_1^{(i)} \otimes \cdots \otimes \boldsymbol{W}_d^{(i)}$ and '+' serves the role of stacking matrices vertically. The methods described in this section will exploit the special structure of this class of workloads to scale more effectively than techniques described in Section 3.3.

3.5.1 A special case: Kronecker product workloads

We begin by considering a special case of the workload in Equation (3.3) that is a *single Kronecker product.* For workloads of this form, we propose optimizing the subworkloads on each attribute individually and then take the Kronecker product of the optimized substrategies to form a strategy for the original workload. We optimize the subworkloads using OPT_0.

Definition 19 ($\mathsf{OPT}_\otimes$)**.** *Given a Kronecker product workload* $\mathbb{W} = \boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d$ *and an optimization oracle* OPT_0*, let* $\mathsf{OPT}_\otimes(\mathbb{W}) = \boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d$ *where* $\boldsymbol{Q}_i = \mathsf{OPT}_0(\boldsymbol{W}_i)$.

Above, OPT_0 may be any strategy optimization routine that consumes an explicitly represented workload and returns a strategy matrix, such as the techniques discussed in the Section 3.3. Since $\mathsf{OPT}_\otimes$ requires solving d small subproblems rather than one large problem, it can be far more efficient than OPT_0 for this class of workloads. This decomposition of the objective function has a well-founded theoretical justification. Namely, if we restrict the solution space to a (single) Kronecker product strategy of the form $\mathbb{Q} = \boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d$, then the error of the workload under $\mathbb{Q}$ decomposes over the factors of the Kronecker products as shown in the following theorem:

Theorem 2 (Error decomposition)**.** *Given a workload* $\mathbb{W} = \boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d$ *and a strategy* $\mathbb{Q} = \boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d$*, the error is:*

$$\|\mathbb{Q}\|_{\mathcal{A}}^2 \left\|\mathbb{W}\mathbb{Q}^+\right\|_F^2 = \prod_{i=1}^{d} \|\boldsymbol{Q}_i\|_{\mathcal{A}}^2 \left\|\boldsymbol{W}_i\boldsymbol{Q}_i^+\right\|_F^2$$

The overall error is minimized when $\boldsymbol{Q}_i$ optimizes $\boldsymbol{W}_i$ for each i, thus it makes sense to optimize each $\boldsymbol{Q}_i$ separately. If we expect the optimal strategy to be a single Kronecker product, then this approach seems quite appealing. However it is possible that there exists a strategy that is *not* a single Kronecker product that offers lower error than the best Kronecker product strategy. The following theorem shows that this is not the case, and gives further justification for the above method, showing that the SVD bound also decomposes over the factors of the Kronecker product.

Theorem 3 (SVD bound decomposition)**.** *Given a workload* $\mathbb{W} = \boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d$*, the SVD bound is:*

$$SVDB(\mathbb{W}) = \prod_{i=1}^{d} SVDB(\boldsymbol{W}_i)$$

If there exist strategies $\boldsymbol{Q}_1, \dots, \boldsymbol{Q}_d$ that achieve the SVD bound for $\boldsymbol{W}_1, \dots, \boldsymbol{W}_d$ and we can find them, then we can construct a Kronecker product strategy $\mathbb{Q} = \boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d$ that achieves the SVD bound for $\mathbb{W} = \boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d$. Since no other strategy can have lower error than the SVD bound, in these situations *the optimal strategy is a Kronecker product.* This gives excellent justification for optimizing over the space of Kronecker products, especially when the SVD bound is tight.

3.5.2 The general case: union of Kronecker product workloads

The approach just described is principled and effective when the workload is a single Kronecker product. We now turn our attention to the more general case where the workload is a union of Kronecker products. Here, the right approach is less clear. We define three approaches for optimizing implicit workloads in the form of Equation (3.3). Each approach restricts the strategy to a different region of the full strategy space for which optimization is tractable. The first computes a strategy consisting of a single Kronecker product; it generalizes $\mathsf{OPT}_{\otimes}$. The second, OPT_{+}, can generate strategies consisting of unions of Kronecker products. The third, $\mathsf{OPT}_{\mathsf{M}}$, generates a strategy of weighted marginal queries. The best approach to use will generally depend on the workload, and we will provide some practical guidance and intuition to understand the situations in which each method works best.

Single-product output strategy For a workload in the form of Equation (3.3), if we restrict the optimization problem to a single Kronecker product strategy, then the objective function decomposes as follows:

Theorem 4. *Given workload $\mathbb{W} = w_1\mathbb{W}_1 + \ldots + w_k\mathbb{W}_k$ and strategy $\mathbb{Q} = \boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d$, workload error is:*

$$\begin{aligned} \|\mathbb{Q}\|_{\mathcal{A}}^2 \left\|\mathbb{W}\mathbb{Q}^+\right\|_F^2 &= \|\mathbb{Q}\|_{\mathcal{A}}^2 \sum_{j=1}^{k} w_j^2 \left\|\mathbb{W}_j\mathbb{Q}^+\right\|_F^2 \\ &= \sum_{j=1}^{k} w_j^2 \prod_{i=1}^{d} \|\boldsymbol{Q}_i\|_{\mathcal{A}}^2 \left\|\boldsymbol{W}_i^{(j)}\boldsymbol{Q}_i^+\right\|_F^2 \end{aligned} \tag{3.4}$$

This leads to the following optimization problem:

Definition 20 (Generalized $\mathsf{OPT}_{\otimes}$)**.** *Given a workload $\mathbb{W} = w_1\mathbb{W}_1 + \ldots + w_k\mathbb{W}_k$, let $\mathsf{OPT}_{\otimes}(\mathbb{W}) = \boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d$ where:*

$$(\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_d) = \underset{\boldsymbol{Q}_1,\ldots,\boldsymbol{Q}_d}{\mathit{minimize}} \quad \sum_{j=1}^{k} w_j^2 \prod_{i=1}^{d} \|\boldsymbol{Q}_i\|_{\mathcal{A}}^2 \left\|\boldsymbol{W}_i^{(j)}\boldsymbol{Q}_i^+\right\|_F^2$$

When $k = 1$, the solution to the problem in Definition 20 is given in Definition 19, so we use matching notation and allow the $\mathsf{OPT}_{\otimes}$ operator to accept a single product or a union of products.

We can attempt to solve this problem by building on the optimization oracles designed in the previous section. In particular, suppose we have a black box optimization oracle $\mathsf{OPT}_0(\boldsymbol{W})$ that accepts an explicitly represented workload and gives back an explicitly represented strategy with low (ideally minimal) error on that workload. Then we use a block method that cyclically optimizes $\boldsymbol{Q}_1, \dots, \boldsymbol{Q}_d$ until convergence. We begin by initializing $\boldsymbol{Q}_i = \boldsymbol{I}$ for all i. We then optimize one $\boldsymbol{Q}_i$ at a time, fixing the other $\boldsymbol{Q}_{i'} \neq \boldsymbol{Q}_i$ using OPT_0 on a carefully constructed surrogate workload $\hat{\boldsymbol{W}}$ (Equation (3.5)) that has the property that the error of any strategy $\boldsymbol{Q}_i$ on $\hat{\boldsymbol{W}}$ is the same as the error of $\mathbb{Q}$ on $\mathbb{W}$. Hence, the correct objective is being minimized.

$$\hat{\boldsymbol{W}}_i = \begin{bmatrix} c_1 \boldsymbol{W}_i^{(1)} \\ \vdots \\ c_k \boldsymbol{W}_i^{(k)} \end{bmatrix} \qquad c_j = w_j \prod_{i' \neq i} \|\boldsymbol{Q}_{i'}\|_{\mathcal{A}} \left\| \boldsymbol{W}_{i'}^{(j)} \boldsymbol{Q}_{i'}^{+} \right\|_F \tag{3.5}$$

The cost of running this optimization procedure is determined by the cost of computing $\hat{\boldsymbol{W}}_i^T \hat{\boldsymbol{W}}_i$ and the cost of optimizing it, which takes $O(n_i^2(p_i + k))$ and $O(n_i^2 p_i \cdot \#\text{ITER})$ time respectively, assuming each $(\boldsymbol{W}^T\boldsymbol{W})_i^{(j)}$ has been precomputed. As before, this method scales to arbitrarily large domains as long as the domain size of the subproblems allows OPT_0 to be efficient.

Union-of-Kronecker output strategy For certain workloads, restricting to solutions consisting of a single Kronecker product, as $\mathsf{OPT}_{\otimes}$ does, excludes good strategies, as demonstrated in Example 2.

Example 2. *Consider the workload* $\mathbb{W} = \mathbb{W}_1 + \mathbb{W}_2$ *where* $\mathbb{W}_1 = \boldsymbol{P} \otimes \boldsymbol{T}$ *and* $\mathbb{W}_2 = \boldsymbol{T} \otimes \boldsymbol{P}$ *on a 2-dimensional domain of size* 100×100. *Running* $\mathsf{OPT}_{\otimes}$ *on this workload leads to an optimized strategy of the form* $\mathbb{Q} = \boldsymbol{Q}_1 \otimes \boldsymbol{Q}_2$. *The expected error of this strategy* *is* 33385*, which is much higher than it should be for such a simple workload. The poor expected error can be explained by the fact that to support the workload, both* $\boldsymbol{Q}_1$ *and* $\boldsymbol{Q}_2$ *have to be full rank. This means* $\mathbb{Q}$ *has to include at least* 100^2 *queries, even though* $\mathbb{W}$ *only contains* 200 *queries.*

A better alternative would be to optimize $\mathbb{W}_1$ *and* $\mathbb{W}_2$ *separately using* $\mathsf{OPT}_{\otimes}$*. Doing this we would end up with a strategy* $\mathbb{Q} = \mathbb{Q}_1 + \mathbb{Q}_2$*, where* $\mathbb{Q}_1$ *optimizes* $\mathbb{W}_1$ *and* $\mathbb{Q}_2$ *optimizes* $\mathbb{W}_2$*. The resulting strategy is much smaller because* $\mathsf{OPT}_{\otimes}(\mathbb{Q}_1) = \mathsf{OPT}_0(\boldsymbol{P}) \otimes \mathsf{OPT}_0(\boldsymbol{T})$*, and* $\mathsf{OPT}_0(\boldsymbol{T}) = \boldsymbol{T}$*. In fact, it only contains* 212 *queries and attains an expected error of* 14252*, which is a* $2.34\times$ *improvement.*

Based on this example, we would like a principled approach to optimize over the space of strategies that are a union of Kronecker products. Unfortunately, computing the workload error exactly for a strategy of this form is intractable, as the pseudo inverse may not be a union of Kronecker products. This makes optimization over this space of strategies challenging. We thus propose the following heuristic optimization routine inspired by Example 2. This optimization routine individually optimizes each subworkload $\mathbb{W}_j$ using $\mathsf{OPT}_{\otimes}$, and then combines the strategies all together to form a single strategy. It simply requires calling $\mathsf{OPT}_{\otimes}$ a number of times and computing appropriate weights for each optimized strategy.

Definition 21 (OPT_+)**.** *Given a workload* $\mathbb{W} = w_1\mathbb{W}_1 + \dots + w_k\mathbb{W}_k$*, let* $\mathsf{OPT}_+(\mathbb{W}) = c_1\mathbb{Q}_1 + \dots + c_k\mathbb{Q}_k$ *where* $\mathbb{Q}_j = \mathsf{OPT}_{\otimes}(\mathbb{W}_j)$ *and*

$$c_j \propto \frac{1}{\|\mathbb{Q}_j\|_{\mathcal{A}}} \begin{cases} \sqrt[3]{2E_j} & \text{if } \mathcal{A} = \mathcal{L} \\ \sqrt[4]{E_j} & \text{if } \mathcal{A} = \mathcal{G} \end{cases}$$

for $E_j = w_j^2 \|\mathbb{Q}_j\|_{\mathcal{A}}^2 \left\|\mathbb{W}_j\mathbb{Q}_j^+\right\|_F^2$.

Above, we assume that $\mathbb{Q}_j$ will be used to answer $\mathbb{W}_j$, and c_j is the weight on $\mathbb{Q}_j$: it corresponds the portion of the privacy budget that will be spent to answer those queries. It is chosen to minimize total workload error. Specifically, if we allocate c_j budget to answer $\mathbb{Q}_j$, then the error will be E_j / c_j^2. Thus, the choice of c_j above is based on minimizing $\sum E_j / c_j^2$ subject to the constraint $\sum |c_j| = 1$ (for Laplace noise) and $\sum c_j^2 = 1$ (for Gaussian noise). We can solve this minimization problem exactly, in closed from, using the method of Lagrange multipliers.

We remark that in the definition above, $\mathbb{W}$ is split up into k subworkloads $\mathbb{W}_1, \dots, \mathbb{W}_k$. Each subworkload $\mathbb{W}_j$ is assumed to be a single Kronecker product, but the optimization routine is still well defined even if $\mathbb{W}_j$ is a union of Kronecker products. This opens up a nice opportunity: to group the subworkloads into clusters which will be optimized together with $\mathsf{OPT}_{\otimes}$. Intuitively, if two subworkloads are similar, it may make sense to group them together to optimize collectively. We do not provide an automated way to group subworkloads in this thesis. This is a nontrivial problem in general, and is out of scope for this thesis. A domain expert can work out good clusterings on a case-by-case basis, or they can simply use the default clustering (one Kronecker product per cluster).

3.6 Implicit representations for marginal queries

In the previous section we described $\mathsf{OPT}_{\otimes}$ and OPT_{+}, two methods for optimizing implicitly represented conjunctive linear query workloads. These methods differ primarily in the space of strategies they search over. Our final optimization method, $\mathsf{OPT}_{\mathsf{M}}$, optimizes over the space of marginal query matrices, and offers a preferable alternative to $\mathsf{OPT}_{\otimes}$ and OPT_{+} in some settings. In order to develop these ideas formally, we must introduce substantial new notation to enable us to work with marginal query matrices and related objects. In this section, we propose an implicit representation for marginal query sets that is even more compact than our other representation for general conjunctive linear query sets. We further show that these matrices can be operated on efficiently, allowing us to solve the strategy optimization problem for large multi-dimensional domains.

A marginal query matrix is a special case of a union of Kronecker products, where each Kronecker product encodes the queries to compute a single marginal. Specifically, all the factors of a marginal query matrix are either $\boldsymbol{I}$ or $\boldsymbol{T}$, where $\boldsymbol{I}$ is an $n_i \times n_i$ identity matrix and $\boldsymbol{T}$ is a $1 \times n_i$ matrix of ones. First note that a marginal on a d-dimensional domain can be specified by a subset of elements of $\{1, \dots, d\}$. Hence, there are a total of 2^d possible marginals, and each one can be specified by an element of the set $[2^d] = \{0, \dots, 2^d - 1\}$. The most natural correspondence between these integers and the associated marginals is based on the binary representation of the integer. The query set required to compute the a^{th} marginal would be represented by $\boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d$ where $\boldsymbol{Q}_i = \boldsymbol{I}$ if the i^{th} bit of the binary representation of a is 1 and $\boldsymbol{Q}_i = \boldsymbol{T}$ otherwise. A collection of weighted marginal queries can thus be represented as a vector $\boldsymbol{u}$ containing a weight for each marginal. We refer to this marginal query matrix as $\mathbb{M}(\boldsymbol{u})$, which is defined below.

Definition 22 (Marginal query matrix)**.** *A marginal query matrix* $\mathbb{M}(\boldsymbol{u})$ *is defined by a vector of weights* $\boldsymbol{u} \in \mathbb{R}^{2^d}$ *and is a special case of the query matrix shown in Equation* (3.3) *where* $k = 2^d$, $w_{a+1} = \boldsymbol{u}(a)$, *and*

$$\boldsymbol{W}_i^{(a+1)} = \begin{cases} \boldsymbol{I} & a_i = 1 \\ \boldsymbol{T} & a_i = 0 \end{cases}$$

where $a \in \{0, \dots, 2^d - 1\}$,$i \in \{1, \dots, d\}$,*and* a_i *is the* i^{th} *bit of the binary representation of* a.

For a marginal query matrix, the weight $\boldsymbol{u}(a)$ can be interpreted as the relative *importance* of the a^{th} marginal. The example below provides further clarification on this implicit representation.

Example 3. *The workoad of all* 2*-way marginals on a* 3*-dimensional domain can be expressed as* $\mathbb{M}(\boldsymbol{w})$ *for* $\boldsymbol{w} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$*. The three non-zero entries of* $\boldsymbol{w}$ *appear at indices* 3*,* 5*, and* 6*, which in binary is* 011_2*,* 101_2 *and* 110_2*. Written in expanded form, this workload is:*

$$\mathbb{M}(\boldsymbol{w}) = \begin{bmatrix} 0 & (\boldsymbol{T} & \otimes & \boldsymbol{T} & \otimes & \boldsymbol{T}) \\ 0 & (\boldsymbol{T} & \otimes & \boldsymbol{T} & \otimes & \boldsymbol{I}) \\ 0 & (\boldsymbol{T} & \otimes & \boldsymbol{I} & \otimes & \boldsymbol{T}) \\ 1 & (\boldsymbol{T} & \otimes & \boldsymbol{I} & \otimes & \boldsymbol{I}) \\ 0 & (\boldsymbol{I} & \otimes & \boldsymbol{T} & \otimes & \boldsymbol{T}) \\ 1 & (\boldsymbol{I} & \otimes & \boldsymbol{T} & \otimes & \boldsymbol{I}) \\ 1 & (\boldsymbol{I} & \otimes & \boldsymbol{I} & \otimes & \boldsymbol{T}) \\ 0 & (\boldsymbol{I} & \otimes & \boldsymbol{I} & \otimes & \boldsymbol{I}) \end{bmatrix} \equiv \begin{bmatrix} \boldsymbol{T} & \otimes & \boldsymbol{I} & \otimes & \boldsymbol{I} \\ \boldsymbol{I} & \otimes & \boldsymbol{T} & \otimes & \boldsymbol{I} \\ \boldsymbol{I} & \otimes & \boldsymbol{I} & \otimes & \boldsymbol{T} \end{bmatrix}$$

As shown in Proposition 13, it is particularly simple to reason about the sensitivity of a marginal query matrix.

Proposition 13. *The sensitivity of a marginal query matrix* $\mathbb{M}(\boldsymbol{u})$ *is:*

$$\|\mathbb{M}(\boldsymbol{u})\|_{\mathcal{L}} = \|\boldsymbol{u}\|_1 \qquad\qquad \|\mathbb{M}(\boldsymbol{u})\|_{\mathcal{G}} = \|\boldsymbol{u}\|_2$$

Moving forward, it is convenient to work with the Gram matrix representation of the marginal query matrix instead. As shown below, there is a simple correspondence between the two:

Proposition 14 (Marginal Gram matrix)**.** *Let* $\mathbb{Q} = \mathbb{M}(\boldsymbol{u})$ *be a marginal query matrix. Then the correpsonding marginal Gram matrix is* $\mathbb{Q}^\top \mathbb{Q} = \mathbb{G}(\boldsymbol{u}^2)$*, where* $\boldsymbol{u}^2$ *is the element-wise square of* $\boldsymbol{u}$*, and:*

$$\mathbb{G}(\boldsymbol{v}) = \sum_{a=0}^{2^d-1} \boldsymbol{v}(a)\mathbb{C}(a), \qquad\qquad \mathbb{C}(a) = \bigotimes_{i=1}^{d} [\mathbf{1}(a_i = 0) + \boldsymbol{I}(a_i = 1)],$$

In the proposition above, the term $\mathbf{1}(a_i = 0) + \boldsymbol{I}(a_i = 1)$ is shorthand notation for $\mathbf{1}$ if $a_i = 0$ and $\boldsymbol{I}$ if $a_i = 1$. Both $\mathbf{1}$ and $\boldsymbol{I}$ are $n_i \times n_i$ matrices, corresponding to the matrix of all ones, and the identity matrix respectively. We will use this notation frequently in the section, so it is important to understand the exact meaning. Another important object that will appear repeatedly throughout this section is the so-called characteristic vector[3], which is defined below:

Definition 23 (Characteristic Vector)**.** *The characteristic vector $\boldsymbol{c} \in \mathbb{R}^{2^d}$ is defined so that each entry $\boldsymbol{c}(a)$ equals the number of entries in the $(\neg a)^{th}$ marginal*

$$\boldsymbol{c}(a) = \prod_{i=1}^{d} n_i(a_i = 0) + 1(a_i = 1)$$

The term $\neg a$ in the definition above is the bitwise negation of a, and it is obtained by flipping each of the d bits of the integer a. We will rely heavily on this type of bitwise manipulation in this section to reason about the behavior of marginal Gram matrices.

Now that we have introduced the necessary notation for marginal query and Gram matrices, we are ready to show how to perform important matrix operations while respecting the implicit representation. We begin with Theorem 5, which shows that marginal Gram matrices interact nicely under matrix multiplication.

Theorem 5 (Multiplication of Marginal Gram Matrices)**.** *For any $a, b \in [2^d]$,*

$$\mathbb{C}(a)\mathbb{C}(b) = \boldsymbol{c}(a|b)\mathbb{C}(a\&b)$$

where $a|b$ denotes "bitwise or", $a\&b$ denotes "bitwise and", and $\boldsymbol{c}$ is the characteristic vector. Moreover, for any $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{2^d}$

[3]this is not to be confused with the eigenvector.

$$\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v}) = \mathbb{G}(\boldsymbol{X}(\boldsymbol{u})\boldsymbol{v})$$

where $\boldsymbol{X}(\boldsymbol{u})$ is a $2^d \times 2^d$ triangular matrix with entries $\boldsymbol{X}(\boldsymbol{u})(k,b) = \sum_{a:a\&b=k} \boldsymbol{u}(a)\boldsymbol{c}(a|b)$.

Proof. First observe how the matrices $\boldsymbol{I}$ and $\mathbf{1}$ interact under matrix multiplication:

$$\boldsymbol{I}\boldsymbol{I} = \boldsymbol{I} \qquad \boldsymbol{I}\mathbf{1} = \mathbf{1} \qquad \mathbf{1}\boldsymbol{I} = \mathbf{1} \qquad \mathbf{1}\mathbf{1} = n_i\mathbf{1}$$

Now consider the product $\mathbb{C}(a)\mathbb{C}(b)$ which is simplified using Kronecker product identities, logical rules, and bitwise manipulation.

$$\begin{aligned}
&= \bigotimes_{i=1}^{d} [\mathbf{1}(a_i = 0) + \boldsymbol{I}(a_i = 1)][\mathbf{1}(b_i = 0) + \boldsymbol{I}(b_i = 1)] \\
&= \prod_{i=1}^{d} [n_i(a_i = 0 \text{ and } b_i = 0) + 1(a_i = 1 \text{ or } b_i = 1)] \\
&\quad \bigotimes_{i=1}^{d} [\mathbf{1}(a_i = 0 \text{ or } b_i = 0) + \boldsymbol{I}(a_i = 1 \text{ and } b_i = 1)] \\
&= \prod_{i=1}^{d} [n_i((a|b)_i = 0) + 1((a|b)_i = 1)] \bigotimes_{i=1}^{d} [\mathbf{1}((a\&b)_i = 0) + \boldsymbol{I}((a\&b)_i = 1)] \\
&= \boldsymbol{c}(a|b)\mathbb{C}(a\&b)
\end{aligned}$$

Now let $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{2^d}$ and consider the following product:

$$\begin{aligned}
\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v}) &= \Big(\sum_a \boldsymbol{u}(a)\mathbb{C}(a)\Big)\Big(\sum_b \boldsymbol{v}(b)\mathbb{C}(b)\Big) \\
&= \sum_{a,b} \boldsymbol{u}(a)\boldsymbol{v}(b)\mathbb{C}(a)\mathbb{C}(b) \\
&= \sum_{a,b} \boldsymbol{u}(a)\boldsymbol{v}(b)\boldsymbol{c}(a|b)\mathbb{C}(a\&b)
\end{aligned}$$

Observe that $\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v}) = \mathbb{G}(\boldsymbol{w})$ where

$$\boldsymbol{w}(k) = \sum_{a \& b = k} \boldsymbol{u}(a)\boldsymbol{v}(b)\boldsymbol{c}(a|b)$$

The relationship between $\boldsymbol{w}$ and $\boldsymbol{v}$ is clearly linear, and by carefully inspecting the expression one can see that $\boldsymbol{w} = \boldsymbol{X}(\boldsymbol{u})\boldsymbol{v}$ where $\boldsymbol{X}(\boldsymbol{u})(k,b) = \sum_{a:a\&b=k} \boldsymbol{u}(a)\boldsymbol{c}(a|b)$. $\boldsymbol{X}(\boldsymbol{u})$ is an upper triangular matrix because $k = a\&b$, and $a\&b \leq b$ for all a. $\square$

Theorem 5 allows us to efficiently multiply two matrices while maintaining the compact implicit representation. Additionally, it follows immediately from the proof of Theorem 5 that $\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v}) = \mathbb{G}(\boldsymbol{v})\mathbb{G}(\boldsymbol{u})$ — i.e., matrix multiplication is commutative. We can apply Theorem 5 to efficiently find the inverse or generalized inverse of $\mathbb{G}(\boldsymbol{u})$ as well.

Theorem 6 (Inverse of Marginal Gram Matrices)**.** *Let $\boldsymbol{X}(\boldsymbol{u})$ be the matrix defined in Theorem 5. If $\boldsymbol{X}(\boldsymbol{u})$ is invertible, then $\mathbb{G}(\boldsymbol{u})$ is invertible with inverse:*

$$\mathbb{G}^{-1}(\boldsymbol{u}) = \mathbb{G}(\boldsymbol{X}^{-1}(\boldsymbol{u})\boldsymbol{z})$$

where $\boldsymbol{z}(2^d - 1) = 1$ and $\boldsymbol{z}(a) = 0$ for all other a. Moreover, if $\boldsymbol{X}^g(\boldsymbol{u})$ is a generalized inverse of $\boldsymbol{X}(\boldsymbol{u})$, then a generalized inverse of $\mathbb{G}(\boldsymbol{u})$ is given by:

$$\mathbb{G}^g(\boldsymbol{u}) = \mathbb{G}(\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{u})$$

.

Proof. First note that $\mathbb{G}(\boldsymbol{z}) = \mathbb{I}$ (the identity matrix). By Theorem 5,

$$\begin{aligned}\mathbb{G}(\boldsymbol{u})\mathbb{G}^{-1}(\boldsymbol{u}) &= \mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{X}^{-1}(\boldsymbol{u})\boldsymbol{z}) \\ &= \mathbb{G}(\boldsymbol{X}(\boldsymbol{u})\boldsymbol{X}^{-1}(\boldsymbol{u})\boldsymbol{z}) \\ &= \mathbb{G}(\boldsymbol{I}\boldsymbol{z}) = \mathbb{G}(\boldsymbol{z}) = \mathbb{I}\end{aligned}$$

This proves the first part of the theorem statement. For the second part, note that if $\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v})\mathbb{G}(\boldsymbol{u}) = \mathbb{G}(\boldsymbol{u})$, then $\mathbb{G}(\boldsymbol{v})$ is a generalized inverse of $\mathbb{G}(\boldsymbol{u})$. Using $\boldsymbol{v} = \boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{u}$, we have

$$\begin{aligned}
\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v})\mathbb{G}(\boldsymbol{u}) &= \mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v}) \\
&= \mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{u}) \\
&= \mathbb{G}(\boldsymbol{X}(\boldsymbol{u})\boldsymbol{X}(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{u}) \\
&= \mathbb{G}(\boldsymbol{X}(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{X}(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{u}) \\
&= \mathbb{G}(\boldsymbol{I}\boldsymbol{X}(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{u}) \\
&= \mathbb{G}(\boldsymbol{I}\boldsymbol{u}) = \mathbb{G}(\boldsymbol{u})
\end{aligned}$$

Thus, $\mathbb{G}(\boldsymbol{v})$ is a generalized inverse as desired. This completes the proof. □

Because $\boldsymbol{X}(\boldsymbol{u})$ is a triangular matrix, we can compute $\boldsymbol{X}^{-1}(\boldsymbol{u})\boldsymbol{z}$ efficiently in $O(4^d)$ time using back-substitution (quadratic in the size of $\boldsymbol{z}$). Note that $\mathbb{G}(\boldsymbol{u})$ and $\boldsymbol{X}(\boldsymbol{u})$ are invertible if and only if $\boldsymbol{u}(2^d - 1) > 0$. The generalized inverse result holds even for non-invertible matrices. This result is slightly more complicated, but is important because in the most common case where the underlying workload contains low-dimensional marginal queries, $\mathbb{G}$ will be singular.

As we show in Theorem 7, we can readily obtain the eigenvectors and eigenvalues of marginal Gram matrices, which will be useful for efficiently computing the SVD Bound for marginal query workloads. Recall that $\boldsymbol{v}$ is an eigenvector with corresponding eigenvalue λ if $\mathbb{G}(\boldsymbol{w})\boldsymbol{v} = \lambda\boldsymbol{v}$ for some real-valued λ. We use the term *eigenmatrix* to refer to a matrix where each column is an eigenvector that shares the same eigenvalue.

Theorem 7 (Eigenvectors and Eigenvalues of Marginal Gram Matrices)**.** *Let $a \in [2^d]$ and let*

$$\mathbb{V}(a) = \bigotimes_{i=1}^{d} (a_i = 0)\boldsymbol{T} + (a_i = 1)(\boldsymbol{1} - n_i\boldsymbol{I})$$

For any $b \in [2^d]$, $\mathbb{V}(a)$ is an eigenmatrix of $\mathbb{C}(b)$ with corresponding eigenvalue $\boldsymbol{\lambda}(a) = \boldsymbol{c}(b)$ if $a\&b = a$ and $\boldsymbol{\lambda}(a) = 0$ otherwise. Moreover, for any $\boldsymbol{w} \in \mathbb{R}^{2^d}$, $\mathbb{V}(a)$ is an eigenmatrix of $\mathbb{G}(\boldsymbol{w})$ with corresponding eigenvalue $\boldsymbol{\kappa}(a) = \sum_{b:a\&b=a} \boldsymbol{w}(b)\boldsymbol{c}(b)$. That is,

$$\mathbb{C}(b)\mathbb{V}(a) = \boldsymbol{\lambda}(a)\mathbb{V}(a) \qquad\qquad \mathbb{G}(\boldsymbol{w})\mathbb{V}(a) = \boldsymbol{\kappa}(a)\mathbb{V}(a)$$

Proof. Recall that $\mathbb{C}(b) = \bigotimes_{i=1}^{d}[\mathbf{1}(b_i = 0) + \boldsymbol{I}(b_i = 1)]$ and $\boldsymbol{c}(k) = \prod_{i=1}^{d}[n_i(k_i = 0) + 1(k_i = 1)]$. The proof follows from direct calculation:

$$\begin{aligned}
\mathbb{C}(b)\mathbb{V}(a) &= \bigotimes_{i=1}^{d}[(b_i = 0)\mathbf{1} + (b_i = 1)\boldsymbol{I}] \bigotimes_{i=1}^{d}[(a_i = 0)\boldsymbol{T} + (a_i = 1)(\mathbf{1} - n_i\boldsymbol{I})] \\
&= \bigotimes_{i=1}^{d}[(b_i = 0)\mathbf{1} + (b_i = 1)\boldsymbol{I}][(a_i = 0)\boldsymbol{T} + (a_i = 1)(\mathbf{1} - n_i\boldsymbol{I})] \\
&= \bigotimes_{i=1}^{d}[(a_i = 0 \text{ and } b_i = 0)n_i\boldsymbol{T} + (a_i = 0 \text{ and } b_i = 1)\boldsymbol{T} \\
&+ (a_i = 1 \text{ and } b_i = 0)\mathbf{0} + (a_i = 1 \text{ and } b_i = 1)(\mathbf{1} - n_i\boldsymbol{I})] \\
&= \begin{cases} \prod_{i=1}^{d} n_i(b_i = 0) + 1(b_i = 1) \bigotimes_{i=1}^{d}[(a_i = 0)\boldsymbol{T} + (a_i = 1)(\mathbf{1} - n_i\boldsymbol{I})] & a\&b = a \\ \mathbf{0} & \textit{otherwise} \end{cases} \\
&= \begin{cases} \boldsymbol{c}(b)\mathbb{V}(a) & a\&b = a \\ 0\mathbb{V}(a) & \textit{otherwise} \end{cases} \\
&= \boldsymbol{\lambda}(a)\mathbb{V}(a)
\end{aligned}$$

This completes the first part of the proof. For the second part, we have:

$$
\begin{aligned}
\mathbb{G}(\boldsymbol{w})\mathbb{V}(a) &= \sum_{b} \boldsymbol{w}(b)\mathbb{C}(b)\mathbb{V}(a) \\
&= \sum_{b} \boldsymbol{w}(b)\boldsymbol{\lambda}(a)\mathbb{V}(a) \\
&= \sum_{b:a\&b=a} \boldsymbol{w}(b)C(b)\mathbb{V}(a) \\
&= \boldsymbol{\kappa}(a)\mathbb{V}(a)
\end{aligned}
$$

□

Interestingly, the eigenmatrices (and hence the eigenvectors) are the same for all marginal Gram matrices $\mathbb{G}(\boldsymbol{w})$. Furthermore, the corresponding eigenvalues have a very simple (linear) dependence on the weights $\boldsymbol{w}$, and as a result, there is a triangular matrix $\boldsymbol{Y}$ such that $\boldsymbol{\kappa} = \boldsymbol{Y}\boldsymbol{w}$.

3.7 Optimizing conjunctive linear query workloads with marginal query strategies

In this section, we describe $\mathsf{OPT}_{\mathsf{M}}$, an optimization operator that consumes a conjunctive linear query workload $\mathbb{W}$ and returns a marginal query strategy $\mathbb{Q} = \mathbb{M}(\boldsymbol{\theta})$.[4] Theorem 8 is the first key to our derivation of $\mathsf{OPT}_{\mathsf{M}}$. Intuitively, it states that for any conjunctive linear query workload $\mathbb{W}$, there is a marginal Gram matrix $\mathbb{G}(\boldsymbol{w})$ that is equivalent to $\mathbb{W}^{\top}\mathbb{W}$ for the purposes of optimization.

Theorem 8 (Marginal approximation of conjunctive linear query workload)**.** *For any conjunctive linear query workload* $\mathbb{W} = w_1\mathbb{W}_1 + \cdots + w_k\mathbb{W}_k$*, there is a marginal Gram matrix* $\mathbb{G}(\boldsymbol{w})$ [5] *such that* $tr[\mathbb{G}(\boldsymbol{u})\mathbb{W}^{\top}\mathbb{W}] = tr[\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{w})]$ *for all* $\boldsymbol{u}$*.*

[4] We reserve the symbol $\boldsymbol{\theta}$ for *strategies*, and use $\boldsymbol{u}, \boldsymbol{v}$ and $\boldsymbol{w}$ to refer to other marginal Gram matrices.

[5] $\boldsymbol{w}$ is related to, but not equal to $w_1, \ldots, w_k$; $\boldsymbol{w}$ has size $2^d \neq k$.

Proof. Let $\mathbb{V} = \mathbb{W}^T\mathbb{W}$ be the Gram matrix of $\mathbb{W}$: $\mathbb{V} = \sum_{j=1}^k w_j^2 \bigotimes_{i=1}^d \boldsymbol{V}_i^{(j)}$ where $\boldsymbol{V}_i^{(j)} = (\boldsymbol{W}^T\boldsymbol{W})_i^{(j)}$. Now consider the following quantity:

$$\begin{aligned}
tr[\mathbb{G}(\boldsymbol{u})\mathbb{V}] &= tr\Big[\Big(\sum_{a=0}^{2^d-1} \boldsymbol{u}(a) \bigotimes_{i=1}^{d} [\mathbf{1}(a_i = 0) + \boldsymbol{I}(a_i = 1)]\Big)\Big(\sum_{j=1}^{k} w_j^2 \bigotimes_{i=1}^{d} \boldsymbol{V}_i^{(j)}\Big)\Big] \\
&= tr\Big[\sum_{a=0}^{2^d-1} \boldsymbol{u}(a) \sum_{j=1}^{k} w_j^2 \bigotimes_{i=1}^{d} [\mathbf{1}(a_i = 0) + \boldsymbol{I}(a_i = 1)]\boldsymbol{V}_i^{(j)}\Big] \\
&= \sum_{a=0}^{2^d-1} \boldsymbol{u}(a) \sum_{j=1}^{k} w_j^2 \prod_{i=1}^{d} tr[\mathbf{1}\boldsymbol{V}_i^{(j)}](a_i = 0) + tr[\boldsymbol{I}\boldsymbol{V}_i^{(j)}](a_i = 1) \\
&= \sum_{a=0}^{2^d-1} \boldsymbol{u}(a) \sum_{j=1}^{k} w_j^2 \prod_{i=1}^{d} sum[\boldsymbol{V}_i^{(j)}](a_i = 0) + tr[\boldsymbol{V}_i^{(j)}](a_i = 1)
\end{aligned}$$

And observe that it only depends on $\boldsymbol{V}_i^{(j)}$ through its *sum* and *trace*. Thus, we could replace $\boldsymbol{V}_i^{(j)}$ with any matrix that has the same *sum* and *trace*. In particular, we could use $\hat{\boldsymbol{V}}_i^{(j)} = b\boldsymbol{I} + c\mathbf{1}$, where b and c are chosen to satisfy the following linear system:

$$\begin{bmatrix} n_i & n_i \\ n_i & n_i^2 \end{bmatrix} \begin{bmatrix} b \\ c \end{bmatrix} = \begin{bmatrix} tr[\boldsymbol{V}_i^{(j)}] \\ sum[\boldsymbol{V}_i^{(j)}] \end{bmatrix}$$

The matrix $\hat{\mathbb{V}}_j = w_j^2(\hat{\boldsymbol{V}}_1^{(j)} \otimes \cdots \otimes \hat{\boldsymbol{V}}_d^{(j)})$ is nothing more than the Gram matrix for a collection of weighted marginals, or $\mathbb{G}(\boldsymbol{w}_j)$. This is because each factor in the Kronecker product is a weighted sum of $\boldsymbol{I}$ and $\mathbf{1}$, and by using the distributive property it can be converted into the canonical representation.

Thus, the matrix $\sum_j \mathbb{G}(\boldsymbol{w}_j) = \mathbb{G}(\sum_j \boldsymbol{w}_j) = \mathbb{G}(\boldsymbol{w})$ satisfies $tr[\mathbb{G}(\boldsymbol{u})\mathbb{V}] = tr[\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{w})]$ as desired. □

$\mathbb{G}(\boldsymbol{u})$ in Theorem 8 represents $(\mathbb{Q}^\top\mathbb{Q})^+$ in the expected error formula. We know this is a marginal Gram matrix by Proposition 14 and Theorem 6. Theorem 8 allows us to reduce the problem of optimizing an arbitrary conjunctive linear query workload to simply optimizing a marginal query workload, which we can do efficiently. In fact, as we show in Theorem 9, we can efficiently evaluate the matrix mechanism objective for marginal query strategies, which is essential for efficient optimization.

Theorem 9 (Marginal parameterization objective function)**.** *Let* $\mathbb{W} = w_1 \mathbb{W}_1 + \cdots + w_k \mathbb{W}_k$ *be a conjunctive linear query workload and let* $\mathbb{G}(\boldsymbol{w})$ *be the marginal approximation of* $\mathbb{W}^\top \mathbb{W}$ *(as in Theorem 8). For any marginal query strategy* $\mathbb{Q} = \mathbb{M}(\boldsymbol{\theta})$*, the matrix mechanism objective function can be expressed as:*

$$\|\mathbb{Q}\|_{\mathcal{A}}^2 \left\|\mathbb{W}\mathbb{Q}^+\right\|_F^2 = \|\boldsymbol{\theta}\|_{\mathcal{A}}^2 [\mathbf{1}^\top \boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{w}]$$

where $\|\boldsymbol{\theta}\|_{\mathcal{A}}$ *is the sensitivity norm defined in Proposition 13, and* $\boldsymbol{X}$ *is the matrix defined in Theorem 5.*

Proof.

$$\begin{aligned}
\|\mathbb{M}(\boldsymbol{\theta})\|_{\mathcal{A}}^2 \left\|\mathbb{W}\mathbb{M}(\boldsymbol{\theta})^+\right\|_F^2 &= \|\boldsymbol{\theta}\|^2 \left\|\mathbb{W}\mathbb{M}(\boldsymbol{\theta})^+\right\|_F^2 && \text{by Proposition 13} \\
&= \|\boldsymbol{\theta}\|^2\, tr[\mathbb{G}^+(\boldsymbol{\theta}^2)\mathbb{W}^T\mathbb{W}] \\
&= \|\boldsymbol{\theta}\|^2\, tr[\mathbb{G}^+(\boldsymbol{\theta}^2)\mathbb{G}(\boldsymbol{w})] && \text{by Theorem 8} \\
&= \|\boldsymbol{\theta}\|^2\, tr[\mathbb{G}(\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{\theta}^2)\mathbb{G}(\boldsymbol{w})] && \text{by Theorem 6} \\
&= \|\boldsymbol{\theta}\|^2\, tr[\mathbb{G}(\boldsymbol{X}(\boldsymbol{w})\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{\theta}^2)] && \text{by Theorem 5} \\
&= \|\boldsymbol{\theta}\|^2\, tr[\mathbb{G}(\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{X}(\boldsymbol{\theta}^2)\boldsymbol{w})] && \text{by commutativity} \\
&= \|\boldsymbol{\theta}\|^2\, tr[\mathbb{G}(\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{w})] && \text{by constraint} \\
&= \|\boldsymbol{\theta}\|^2 [\mathbf{1}^T \boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{w}]
\end{aligned}$$

□

Theorem 9 shows that we can efficiently calculate the objective function in terms of $\boldsymbol{w}$ and $\boldsymbol{\theta}$, without ever explicitly materializing $\mathbb{G}(\boldsymbol{w})$ or $\mathbb{M}(\boldsymbol{\theta})$. This key idea will allow us to solve the strategy selection problem efficiently. Problem 24 states the main optimization problem that underlies $\mathsf{OPT}_{\mathsf{M}}$, which immediately follows from Theorem 9.

Definition 24 (Marginals parameterization)**.** *Given a conjunctive linear query workload* $\mathbb{W} = w_1\mathbb{W}_1 + \cdots + w_k\mathbb{W}_k$*, let* $\mathsf{OPT}_{\mathsf{M}}(\mathbb{W}) = \mathbb{M}(\boldsymbol{\theta}^*)$ *where*

$$\begin{aligned} \boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \quad & \|\boldsymbol{\theta}\|_{\mathcal{A}}^2 \left[\mathbf{1}^\top \boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{w}\right] \\ \textit{subject to} \quad & \boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{X}(\boldsymbol{\theta}^2)\boldsymbol{w} = \boldsymbol{w} \end{aligned}$$

and $\mathbb{G}(\boldsymbol{w})$ *is the marginal approximation of* $\mathbb{W}^\top\mathbb{W}$ *(as in Theorem 8).*

Above, the constraint ensures that the strategy supports the workload. In practice, this constraint can usually be ignored, and the resulting unconstrained optimization problem can be solved instead. The constraint can then be verified to hold at the end of the optimization. Intuitively, this is because strategies that move closer to the boundary of the constraint will have higher error, so the optimization will never approach it as long as sufficiently small step sizes are taken. We use `scipy.optimize` to solve this problem in practice.

We note that the number of parameters in the above optimization problem is 2^d and that we can evaluate the objective in $O(4^d)$ time (quadratic in the number of parameters). Thus, it is feasible to solve this problem as long as $d \leq 15$. Importantly, this means that the runtime complexity is independent of the domain size of each attribute, so it will take the same amount of time for $n_i = 2$ (binary features), $n_i = 10$, or any other values of n_i.

In addition to being able to efficiently optimize over the space of marginal query strategies, we can also efficiently compute the SVD bound for marginal query workloads.

Theorem 10 gives a simple formula for computing the SVD bound for marginal query workloads.

Theorem 10 (SVD Bound for Marginal Query Workloads)**.** *The SVD bound for a marginal query workload* $\mathbb{W}$ *with Gram matrix* $\mathbb{G}(\boldsymbol{w})$ *is:*

$$SVDB(\mathbb{W}) = \frac{1}{n}\Big(\sum_{a} \boldsymbol{c}(\neg a)\sqrt{\sum_{b: a \& b = a} \boldsymbol{w}(b)\boldsymbol{c}(b)}\Big)^2$$

Proof. From Theorem 7 we know all 2^d unique eigenvalues and corresponding eigenmatrices. The number of rows in each eigenmatrix corresponds to the number of eigenvectors with that eigenvalue. To compute the SVD bound, we need to take the square root of each unique eigenvalue (which is a singular value of $\mathbb{W}$) and multiply that by it's multiplicity, then sum across all unique eigenvalues. Note that the eigenmatrix $\mathbb{V}(a)$ has $\boldsymbol{c}(\neg a)$ rows. Hence, the SVD bound is:

$$\begin{aligned} SVDB(\mathbb{W}) &= \frac{1}{n}\Big(\sum_{a} \boldsymbol{c}(\neg a)\sqrt{\boldsymbol{\kappa}(a)}\Big)^2 \\ &= \frac{1}{n}\Big(\sum_{a} \boldsymbol{c}(\neg a)\sqrt{\sum_{b: a \& b = a} \boldsymbol{w}(b)\boldsymbol{c}(b)}\Big)^2 \end{aligned}$$

□

Additionally, as a byproduct of this analysis, we give a similarly simple formula to *find the optimal marginal query strategy in closed form* in Theorem 11, allowing us to bypass the need for numerical optimization in some settings.

Theorem 11 (Closed form solution to Problem 24)**.** *Let* $\mathbb{W}$ *be a workload with Gram matrix* $\mathbb{G}(\boldsymbol{w})$ *and let* $\boldsymbol{\theta} = \sqrt{\boldsymbol{Y}^{-1}\sqrt{\boldsymbol{Y}\boldsymbol{w}}}$ *(element-wise square root), where* $\boldsymbol{Y}$ *is the* $2^d \times 2^d$ *matrix:*

$$\boldsymbol{Y}(a,b) = \begin{cases} \boldsymbol{c}(b) & a\&b = a \\ 0 & otherwise \end{cases}$$

If $\boldsymbol{\theta}$ contains real-valued entries then the strategy $\mathbb{Q} = \mathbb{M}(\boldsymbol{\theta})$ attains the SVDB bound when $\mathcal{A} = \mathcal{G}$, and is thus an optimal strategy. That is, $\|\mathbb{Q}\|_{\mathcal{G}}^2 \|\mathbb{W}\mathbb{Q}^+\|_F^2 = SVDB(\mathbb{W})$.

Proof. We will prove optimality by showing that $\mathbb{Q} = \mathbb{M}(\boldsymbol{\theta})$ matches the SVD bound. Li et al. showed that the SVD bound is satisfied with equality if $\mathbb{Q}$ and $\mathbb{W}$ share the same singular vectors and the singular values of $\mathbb{Q}$ are the square root of the singular values of $\mathbb{W}$, at least in the case of Gaussian noise. Recall from Theorem 7 we know that all marginal Gram matrices share the same eigenvectors. The unique eigenvalues of $\mathbb{G}(\boldsymbol{w})$ are $\boldsymbol{\kappa} = \boldsymbol{Y}\boldsymbol{w}$. The gram matrix of $\mathbb{Q} = \mathbb{M}(\boldsymbol{\theta})$ is $\mathbb{Q}^T\mathbb{Q} = \mathbb{G}(\boldsymbol{\theta}^2)$. The eigenvalues of this are $\boldsymbol{Y}\boldsymbol{\theta}^2 = \boldsymbol{Y}(\boldsymbol{Y}^{-1}\sqrt{\boldsymbol{Y}\boldsymbol{w}}) = \sqrt{\boldsymbol{Y}\boldsymbol{w}}$. Thus, the eigenvalues are exactly the square root of the eigenvalues of $\mathbb{G}(\boldsymbol{w})$, as desired. This certifies that $\mathbb{Q} = \mathbb{M}(\boldsymbol{\theta})$ matches the SVD bound and is optimal. $\square$

While $\boldsymbol{\theta}$ may sometimes contain imaginary entries, we can always fall back on numerical optimization to solve Problem 24. The formula in Theorem 11 can still be used to initialize the optimization if the imaginary entries of $\boldsymbol{\theta}$ are replaced with zeros. Li and Miklau derived sufficient conditions for the SVD bound to be realizable [59], and marginal query workloads satisfy those sufficient conditions. This implies that the SVD bound should always be attainable for workloads of this form. If the parameters in Theorem 11 contain imaginary entries, this suggests that the optimal strategy is *not* a marginal query strategy. It is an interesting open question to determine what the structure of the optimal strategy is when Theorem 11 does not apply. In practice, even when the SVD bound is not attained exactly by OPT_M, we get very close to it for marginal query workloads, as we show empirically in Table 3.5 of the experiments.

3.8 The $\mathsf{OPT}_{\mathsf{HDMM}}$ strategy selection algorithm

Definition		Operator	Input workload	Output strategy	Complexity
§3.3	Definitions 13 and 15	OPT_0	Explicit matrix $\boldsymbol{W}$	Explicit matrix $\boldsymbol{Q}$	$O(n^3)$
§3.5.1	Definition 19	$\mathsf{OPT}_\otimes$	Kronecker Product	Kronecker product	$O(\sum_{i=1}^d n_i^3)$
§3.5.2	Definition 20	$\mathsf{OPT}_\otimes$	Union of Kronecker Products	Kronecker Product	$O(k\sum_{i=1}^d n_i^3)$
§3.5.2	Definition 21	OPT_+	Union of Kronecker Products	Union of Kronecker Products	$O(k\sum_{i=1}^d n_i^3)$
§3.7	Definition 24	OPT_M	Union of Kronecker Products	Marginal Query Strategy	$O(4^d)$

Table 3.1: Summary of optimization operators: input and output types, and the time complexity of objective/gradient calculations. $n = n_1 \times \cdots \times n_d$ refers to the domain size, and k (where applicable) refers to the number of Kronecker products in the workload.

In this chapter, we proposed four optimization routines: OPT_0, $\mathsf{OPT}_\otimes$, OPT_+, and OPT_M. In this section, we summarize these different approaches, discuss the pros and cons of each one, and propose a meta-optimization algorithm $\mathsf{OPT}_{\mathsf{HDMM}}$ that automatically chooses the best one based on the workload. Table 3.1 summarizes the basic inputs and outputs of each operator. OPT_0 is designed to optimize an explicitly represented workload, and returns an explicitly represented strategy. The other optimization operators all operate in an implicit space however.

The time complexity of OPT_0 is $O(n^3)$ (where n is the domain size), and it generally feasible to run as long as $n \leq 10^4$. The time complexity of $\mathsf{OPT}_\otimes$ and OPT_+ is $O(k \sum n_i^3)$, where k is the number of union terms in the workload, and n_i is the domain size of attribute i. It is generally feasible to run as long as OPT_0 is feasible on each of the individual attributes (i.e., $n_i \leq 10^4$). In contrast to OPT_0, the total domain size for these operators can be arbitrarily large. The time complexity of OPT_M is $O(4^d)$, which interestingly does not depend on the domain size of individual attributes, only the number of attributes. It is generally feasible to run as long as $d \leq 15$.

Each of the operators searches over a different space of strategies, and the best one to use will ultimately depend on the workload. We illustrate the behavior of each optimization operator on the simple workload of all 2-way marginals in Example 4.

This example highlights and summarizes the key differences between $\mathsf{OPT}_{\otimes}$, OPT_{+}, and $\mathsf{OPT}_{\mathsf{M}}$. In this case, $\mathsf{OPT}_{\mathsf{M}}$ is the best, which is not surprising because it is the most suitable for marginal workloads. It achieves this by placing more weight on the queries for larger marginals, and less weight on other queries. When compared to the baseline of using $\mathbb{W}$ as the strategy, $\mathsf{OPT}_{\mathsf{M}}$ achieves lower error on the larger marginals but has higher error on the smaller marginals. As a result, $\mathsf{OPT}_{\mathsf{M}}$ enjoys lower *overall* error than the simple baseline, but suffers higher *max* error. The expected errors reported in Example 4 pertain to TSE from Definition 26.

In general predicting which optimization operator will yield the lowest error strategy requires domain expertise and may be challenging for complex workloads. Since strategy selection is independent of the input data and does not consume the privacy budget, we can just run each optimization operator, keeping the output strategy that offers the smallest expected error. Additionally, since the strategies found by each optimization operator may depend on the initialization, we recommend running several random restarts of each optimization operator, returning the best one.

By default, $\mathsf{OPT}_{\mathsf{HDMM}}$ invokes all three high-dimensional optimization operators $\mathsf{OPT}_{\otimes}$, OPT_{+}, and $\mathsf{OPT}_{\mathsf{M}}$. ($\mathsf{OPT}_{0}$ may also be included for lower-dimensional workloads). For $\mathsf{OPT}_{\otimes}$ and OPT_{+} invoked with the p-Identity strategy we use the following convention for setting the p parameters: if an attribute's subworkload is completely defined in terms of $\boldsymbol{T}$ and $\boldsymbol{I}$, we set $p = 1$ (this is a fairly common case where more expressive strategies do not help), otherwise we set $p = n_i/16$ for each attribute A_i with size n_i.

Example 4 (Optimizing Marginal Query Workload)**.** *Consider the workload containing queries to compute all 2-way marginals on a domain of size* $(2, 5, 50, 100)$*. This workload can be represented as a union of* $\binom{4}{2} = 6$ *Kronecker products. The table below gives the precise representation of this workload, together with the optimized strategies found by* $\mathsf{OPT}_{\otimes}$, OPT_{+}, *and* $\mathsf{OPT}_{\mathsf{M}}$*. All optimized strategies can be expressed in terms of the "Identity"* ($\boldsymbol{I}$) *and "Total"* ($\boldsymbol{T}$) *building blocks.*

	Query Matrix	TSE
$\mathbb{W}$	$\boldsymbol{T} \otimes \boldsymbol{T} \otimes \boldsymbol{I} \otimes \boldsymbol{I}$ $\boldsymbol{T} \otimes \boldsymbol{I} \otimes \boldsymbol{T} \otimes \boldsymbol{I}$ $\boldsymbol{T} \otimes \boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{T}$ $\boldsymbol{I} \otimes \boldsymbol{T} \otimes \boldsymbol{T} \otimes \boldsymbol{I}$ $\boldsymbol{I} \otimes \boldsymbol{T} \otimes \boldsymbol{I} \otimes \boldsymbol{T}$ $\boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{T} \otimes \boldsymbol{T}$	$206,964$
$\mathbb{I}$	$\boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{I}$	$300,000$
$\mathsf{OPT}_{\otimes}(\mathbb{W})$	$\boldsymbol{I} \otimes \boldsymbol{I} \otimes \begin{bmatrix} 0.80\,\boldsymbol{I} \\ 0.20\,\boldsymbol{T} \end{bmatrix} \otimes \begin{bmatrix} 0.82\,\boldsymbol{I} \\ 0.18\,\boldsymbol{T} \end{bmatrix}$	$213,270$
$\mathsf{OPT}_{+}(\mathbb{W})$	*0.39* $\boldsymbol{T} \otimes \boldsymbol{T} \otimes \boldsymbol{I} \otimes \boldsymbol{I}$ *0.18* $\boldsymbol{T} \otimes \boldsymbol{I} \otimes \boldsymbol{T} \otimes \boldsymbol{I}$ *0.14* $\boldsymbol{T} \otimes \boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{T}$ *0.13* $\boldsymbol{I} \otimes \boldsymbol{T} \otimes \boldsymbol{T} \otimes \boldsymbol{I}$ *0.11* $\boldsymbol{I} \otimes \boldsymbol{T} \otimes \boldsymbol{I} \otimes \boldsymbol{T}$ *0.05* $\boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{T} \otimes \boldsymbol{T}$	$85,070$
$\mathsf{OPT}_{\mathsf{M}}(\mathbb{W})$	*0.44* $\boldsymbol{T} \otimes \boldsymbol{T} \otimes \boldsymbol{I} \otimes \boldsymbol{I}$ *0.31* $\boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{T} \otimes \boldsymbol{I}$ *0.25* $\boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{T}$	*62,886*

3.9 Efficient `measure` and `reconstruct`

Now that we have fully described how HDMM solves the `select` subproblem, we are ready to discuss how to run the remainder of the mechanism. To `measure` the selected strategy queries, we must compute the matrix vector product $\boldsymbol{y} = \mathbb{Q}\boldsymbol{p} + \boldsymbol{\xi}$, and to reconstruct the workload query answers, we must compute $\mathbb{W}\mathbb{Q}^{+}\boldsymbol{y}$. With explicitly represented matrices, these computations can be done directly without problem. However, when $\mathbb{W}$ and $\mathbb{Q}$ are too large to represent explicitly, it is no longer obvious how to run the mechanism. A necessary key subroutine to solve these problems is to compute matrix-vector products where the matrix is a Kronecker product, without ever materializing the matrix explicitly.

Algorithm 1 Kronecker Matrix-Vector Product

```
 1: Input: Matrices $\boldsymbol{Q}_1, \dots, \boldsymbol{Q}_d$, vector $\boldsymbol{p}$
 2: Output: Vector $(\boldsymbol{Q}_1 \otimes \dots \otimes \boldsymbol{Q}_d)\boldsymbol{p}$
 3: $m_i, n_i = \text{SHAPE}(\boldsymbol{Q}_i)$
 4: $r = n$
 5: $\boldsymbol{f}_{d+1} = \boldsymbol{p}$
 6: for $i = d, \dots, 1$ do
 7:     $\boldsymbol{Z} = \text{RESHAPE}(\boldsymbol{f}_{i+1}, n_i, r/n_i)$
 8:     $r = r \cdot m_i/n_i$
 9:     $\boldsymbol{f}_i = \text{RESHAPE}(\boldsymbol{Q}_i\boldsymbol{Z}, r, 1)$
10: end for
11: return $\boldsymbol{f}_1$
```

Theorem 12 (Efficient matrix-vector multiplication)**.** *Let $\mathbb{Q} = \boldsymbol{Q}_1 \otimes \dots \otimes \boldsymbol{Q}_d$ and let $\boldsymbol{p}$ be a data vector of compatible shape. Then Algorithm 1 computes the matrix-vector product $\mathbb{Q}\boldsymbol{p}$. Furthermore, if $\boldsymbol{Q}_i \in \mathbb{R}^{n_i \times n_i}$ and $n = \prod n_i$ is the size of $\boldsymbol{p}$ then Algorithm 1 runs in $O(n \sum n_i)$ time.*

Proof. Let $\boldsymbol{y} = \mathbb{Q}\boldsymbol{p}$. Then

$$\begin{aligned}
\boldsymbol{y}(z) &= \sum_{x \in \Omega} \mathbb{Q}(z, x)\boldsymbol{p}(x) \\
&= \sum_{x \in \Omega} \boldsymbol{Q}_1(z_1, x_1) \dots \boldsymbol{Q}_d(z_d, x_d)\boldsymbol{p}(x) \\
&= \sum_{x_1 \in \Omega_1} \boldsymbol{Q}_1(z_1, x_1) \cdots \sum_{x_d \in \Omega_d} \boldsymbol{Q}_d(z_d, x_d)\boldsymbol{p}(x_1, \dots, x_d)
\end{aligned}$$

Now define $\boldsymbol{f}_i$ to be the vector indexed by tuples $(x_1, \dots, x_{i-1}, z_i, \dots, z_d)$ such that $\boldsymbol{f}_{d+1} = \boldsymbol{p}$ and:

$$\boldsymbol{f}_i(x_{1:i-1}, z_{i:d}) = \sum_{x_i \in \Omega_i} \boldsymbol{Q}_i(z_i, x_i)\boldsymbol{f}_{i+1}(x_{1:i}, z_{i+1:d})$$

and observe that $\boldsymbol{y} = \boldsymbol{f}_1$. We can efficiently compute $\boldsymbol{f}_i$ from $\boldsymbol{f}_{i+1}$ by observing that it is essentially computing a matrix-matrix product between the $n_i \times n_i$ matrix $\boldsymbol{Q}_i$ and the $n_i \times n/n_i$ matrix obtained by reorganizing the entries of $\boldsymbol{f}_{i+1}$ into a matrix where rows are indexed by x_i. This can be computed in $O(nn_i)$ time. Thus, the total time required to compute $\boldsymbol{y}$ is $O(n \sum n_i)$ as stated. $\square$

Algorithm 1 is correct even if the factors of $\mathbb{Q}$ are not square, although the time complexity is not as clean when written down. Since all of the strategies found by our optimization routines are either Kronecker products or unions thereof, we can directly apply Algorithm 1 to efficiently implement the `measure` step of HDMM. Note that computing the matrix-vector product for a union of Kronecker products is a trivial extension of Algorithm 1: it simply requires calling Algorithm 1 for each Kronecker product and concatenating the results into a single vector.

We can also use Algorithm 1 to efficiently implement the `reconstruct` step of HDMM. The main challenge is to compute $\mathbb{Q}^{+}\boldsymbol{y}$, or a pseudoinverse of $\mathbb{Q}$ together with a matrix-vector product. This is done slightly differently for each type of strategy:

1. $\mathbb{Q} = \mathsf{OPT}_{\otimes}(\mathbb{W}) = \boldsymbol{Q}_1 \otimes \cdots \times \boldsymbol{Q}_d$. From Proposition 11 we know that $\mathbb{Q}^+ = \boldsymbol{Q}_1^+ \otimes \cdots \otimes \boldsymbol{Q}_d^+$. That is, the pseudoinverse of a Kronecker product is still a Kronecker product. Thus, we can compute $\hat{\boldsymbol{p}} = \mathbb{Q}^+ \boldsymbol{y}$ efficiently using Algorithm 1.

2. $\mathbb{Q} = \mathsf{OPT}_{\mathsf{M}}(\mathbb{W}) = \mathbb{M}(\boldsymbol{\theta})$. From basic linear algebra, we know that $\boldsymbol{Q}^+ = (\boldsymbol{Q}^\top \boldsymbol{Q})^+ \boldsymbol{Q}^\top$ for any matrix $\boldsymbol{Q}$. Applied to this setting, we have $\mathbb{M}^+(\boldsymbol{\theta}) = \mathbb{G}^+(\boldsymbol{\theta}^2)\mathbb{M}^\top(\boldsymbol{\theta})$, since we know $\mathbb{M}^\top\mathbb{M}(\boldsymbol{\theta}) = \mathbb{G}(\boldsymbol{\theta}^2)$ by Proposition 14. From Theorem 6 we know how to compute $\mathbb{G}^+(\boldsymbol{\theta}^2)$ efficiently, and we know that it equals $\mathbb{G}(\boldsymbol{\eta})$ for some $\boldsymbol{\eta}$. We aim to compute $\mathbb{M}^+(\boldsymbol{\theta})\boldsymbol{y} = \mathbb{G}^+(\boldsymbol{\theta}^2)\mathbb{M}^\top(\boldsymbol{\theta})\boldsymbol{y}$. We can easily compute $\boldsymbol{v} = \mathbb{M}^\top(\boldsymbol{\theta})\boldsymbol{y}$ using a sequence of calls to Algorithm 1 by observing that $\mathbb{M}^\top(\boldsymbol{\theta})$ is a just a bunch of Kronecker products *horizontally* stacked together. In a similar fashion, we can compute $\hat{\boldsymbol{p}} = \mathbb{G}^+(\boldsymbol{\theta}^2)\boldsymbol{v}$ because $\mathbb{G}^+(\boldsymbol{\theta}^2)$ is just the sum of a bunch of Kronecker products, which we can handle efficiently with repeated calls to Algorithm 1.

3. $\mathbb{Q} = \mathsf{OPT}_{+}(\mathbb{W}) = c_1\mathbb{Q}_1 + \cdots + c_k\mathbb{Q}_k$. Unfortunately, for a strategy of this form, we do not have a way to efficiently compute $\mathbb{Q}^+\boldsymbol{y}$. While $\mathbb{Q}$ is a union of Kronecker products, the pseudoinverse is not necessarily, and we are not aware of a simple formula for the pseudoinverse of $\mathbb{Q}$ at all. However, we can still produce an unbiased estimate of $\mathbb{W}\boldsymbol{p}$ by using *local least squares*. To do this, we will compute $\mathbb{W}_j\mathbb{Q}_j^+\boldsymbol{y}_j$ for each $j = 1, \dots, k$, where $\boldsymbol{y}_j$ is the answers produced for sub-strategy $\mathbb{Q}_j^+$. Since $\mathbb{W}_j$ and $\mathbb{Q}_j^+$ are both assumed to be Kronecker products, this can be easily achieved using Algorithm 1. While $\mathbb{W}_j\mathbb{Q}_j^+\boldsymbol{y}_j$ is an unbiased estimator for $\mathbb{W}_j\boldsymbol{p}$, the workload query answers will not necessarily be consistent between different j.

3.10 Experimental evaluation

In this section we evaluate the accuracy and scalability of HDMM. We perform a comprehensive comparison of HDMM with a variety of other mechanisms on single- and multi-dimensional workloads, showing that it consistently offers lower error than competitors and works in a broader range of settings than other algorithms. We also evaluate the scalability of key components of HDMM, showing that it is capable of scaling effectively to high-dimensional settings.

In accuracy experiments, we report the Root Mean Squared Error (RMSE), which is defined as $RMSE = \sqrt{\frac{1}{m}\text{TSE}(W, \mathcal{A})}$ for an algorithm $\mathcal{A}$. We compute this value analytically using the formulas from Proposition 9 whenever possible. We separately report results for pure differential privacy with Laplace noise and approximate differential privacy with Gaussian noise. We use $\epsilon = 1.0$ and $\delta = 10^{-6}$ in all experiments, but note that the ratio of errors between two data-independent algorithms remains the same for all values of ϵ and δ.

These experiments are meant to demonstrate that HDMM offers the best accuracy in the *data-independent* regime. It is possible that some data-dependent mechanisms will outperform even the best data-independent mechanism, and this will typically depend on the amount of data available and the privacy budget [46, 95]. Data-independent mechanisms (like HDMM) are generally preferable when there is an abundance of data and/or the privacy budget is not too small, such as the U.S. Census decennial data release [2].

3.10.1 Evaluating OPT_0 on low-dimensional workloads

We begin by studying the effectiveness of OPT_0 in the one-dimensional setting. Specifically, we evaluate the quality of the strategies found by our optimization oracle compared with other *data-independent* mechanisms designed for this setting. It is important to understand the accuracy in the one-dimensional setting well, because

ϵ-differential privacy (Laplace noise)

Workload	Domain	Identity	H2	Privelet	HB	GreedyH	LRM	OPT_0	SVDB
All Range	64	6.63	11.28	10.11	6.63	6.34	7.02	**5.55**	3.22
	256	13.11	16.27	14.87	8.90	9.72	15.73	**8.07**	4.07
	1024	26.15	21.83	20.26	12.82	14.70	-	**11.08**	4.94
	4096	52.27	27.90	26.18	16.19	22.21	-	**14.38**	5.82
Prefix	64	8.06	9.42	9.37	8.06	6.04	7.67	**5.32**	2.89
	256	16.03	13.16	13.09	8.97	9.13	12.64	**7.35**	3.50
	1024	32.02	17.29	17.20	12.87	14.32	15.43	**9.58**	4.11
	4096	64.01	21.77	21.67	14.91	22.40	-	**12.20**	4.74
Width 32 Range	64	8.00	12.02	11.09	8.00	7.32	9.44	**5.88**	2.75
	256	8.00	15.50	13.57	7.41	8.00	25.81	**6.34**	3.26
	1024	8.00	18.98	16.56	9.50	8.00	16.98	**6.41**	3.36
	4096	8.00	22.45	19.58	10.96	8.00	-	**6.46**	3.38
Permuted Range	64	6.63	25.02	18.97	6.63	6.83	7.02	**5.55**	3.22
	256	13.11	66.25	49.09	18.48	13.02	15.73	**8.06**	4.07
	1024	26.15	157.50	117.06	37.07	23.94	-	**11.08**	4.94
	4096	52.27	374.29	277.42	107.83	45.77	-	**14.37**	5.82

Table 3.2: Error of strategies for 1D workloads with $\epsilon = 1.0$.

(ϵ, δ)-differential privacy (Gaussian noise)

Workload	Domain	Identity	H2	Privelet	HB	GreedyH	COA	OPT_0	SVDB
All Range	64	19.82	12.74	11.42	19.82	14.64	**9.73**	**9.73**	9.62
	256	39.18	16.20	14.81	18.80	23.34	**12.26**	**12.26**	12.15
	1024	78.13	19.66	18.24	27.07	36.20	14.89	**14.85**	14.75
	4096	156.14	23.12	21.69	27.92	56.21	17.92	**17.46**	17.38
Prefix	64	24.08	10.64	10.58	24.08	14.04	**8.87**	**8.87**	8.62
	256	47.89	13.11	13.03	18.95	22.11	10.70	**10.66**	10.44
	1024	95.64	15.57	15.49	27.18	35.59	16.29	**12.49**	12.29
	4096	191.21	18.03	17.95	25.72	56.70	26.50	**14.32**	14.15
Width 32 Range	64	23.90	13.57	12.52	23.90	17.30	8.79	**8.74**	8.23
	256	23.90	15.44	13.52	15.65	23.90	12.24	**9.93**	9.73
	1024	23.90	17.10	14.92	20.08	23.90	16.00	**10.08**	10.02
	4096	23.90	18.60	16.22	18.90	23.90	18.38	**10.11**	10.09
Permuted Range	64	19.82	28.26	21.42	19.82	16.13	**9.73**	**9.73**	9.62
	256	39.18	65.97	48.88	39.04	35.22	**12.26**	**12.26**	12.15
	1024	78.13	141.86	105.44	78.30	60.60	14.89	**14.85**	14.75
	4096	156.14	310.11	229.85	185.98	118.03	17.92	**17.45**	17.38

Table 3.3: Error of strategies for 1D workloads with $\epsilon = 1.0$ and $\delta = 10^{-6}$.

OPT_0 is used as a sub-routine for the higher-dimensional optimization operators $\mathsf{OPT}_\otimes$ and OPT_+.

Workloads We consider four different workloads: All Range, Prefix, Width 32 Range, and Permuted Range, each defined over domain sizes ranging from 64 to 4096. All Range contains every possible range query over the specified domain; Prefix contains range queries defining an empirical CDF; Width 32 Range contains all range queries of width 32. While the first three workloads are subsets of range queries, the last workload, Permuted Range, is the result of right-multiplying the workload of all range queries by a random permutation matrix. Many proposed strategies have targeted workloads of range queries and tend to work fairly well on subsets of range queries. Permuted Range poses a challenge because the structure of the workload is hidden by the permutation, requiring a truly adaptive method to find a good strategy. Note the large size of some of these workloads: All Range and Permuted Range have $\frac{n(n+1)}{2}$ queries. For large n it is infeasible to write down $\boldsymbol{W}$ in matrix form, but we can still compute the expected error since it only depends on the workload through its Gram matrix, $\boldsymbol{W}^\top \boldsymbol{W}$, which is $n \times n$ and has special structure, allowing it to be computed directly without materializing $\boldsymbol{W}$.

Mechanisms We consider 8 competing mechanisms: Identity, Laplace, Gaussian, LRM [108], COA [107], H2 [47], HB [83], Privelet [100], and GreedyH [56]. The first five mechanisms are general purpose mechanisms, designed to support virtually any workload. The last four mechanisms were specifically designed to offer low error on range query workloads. We also report SVDB to understand the gap between the error of the computed strategies and the best lower bound on error we have (via the SVD bound).

Results and Findings Table 3.2 and Table 3.3 report the error of various mechanisms in each setting, for both Laplace and Gaussian noise respectively. We remind the reader that these values do not depend on the true data $\boldsymbol{p}$, and thus they hold for all $\boldsymbol{p}$. We report numbers for fixed $\epsilon = 1.0$ and $\delta = 10^{-6}$, but we note that these privacy parameters only impact the error by a constant factor, and hence the relationship between the errors of every pair of mechanisms remains the same for all (ϵ, δ). We have four main findings from these results, enumerated below:

1. OPT_0 offers lower error than all competitors in all settings, and the magnitude of the improvement offered by HDMM (over the next best competitor) is as large as 3.18 for Laplace noise (on Permuted Range) and 1.61 for Gaussian noise (on Width 32 Range). Interestingly, OPT_0 offers lower error than H2, HB, Privelet, and GreedyH on range query workloads, even though these four mechanisms were designed specifically for range queries. In additional, the second best method after OPT_0 differs in each setting, which shows that some competing algorithms have specialized capabilities that allow them to perform well in some settings, while HDMM performs well in a variety of settings as it does not make strict assumptions about the workload.

2. OPT_0 gets within a factor of 2.57 of the SVD bound for Laplace noise and 1.01 of the SVD bound for Gaussian noise on every tested workload. The gap between OPT_0 and SVDB is quite small for Gaussian noise, suggesting that OPT_0 is finding the best possible strategy. Note that COA also finds a optimal strategy in many of the settings, but it fails on the Prefix and Width 32 Range workloads when $n \geq 1024$. Thus, even though it is solving the same problem underlying OPT_0 in theory, the implementation is not as robust as ours. The gap between OPT_0 and SVDB is larger for Laplace noise, however, and it is unclear if this gap is primarily due to looseness of the SVD bound or suboptimality of the strategy. Nevertheless, even with Laplace noise the ratio between OPT_0 and SVDB is at most 2.57.

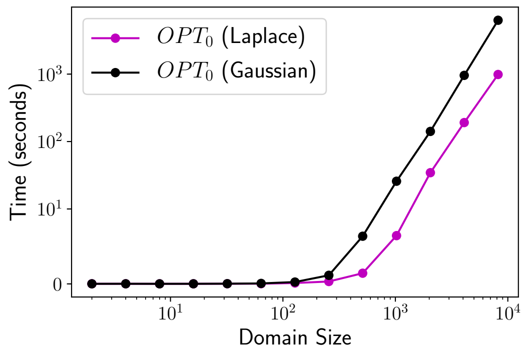


Figure 3.2: Time required to run OPT_0 for 100 iterations on the AllRange workload for increasing domain sizes.

3. The error of OPT_0 (and COA for (ϵ, δ) privacy) is the same on the All Range and Permuted Range workloads. Permuting the workload doesn't impact achievable error or our optimization algorithm in any meaningful way. However, many of the methods we compared against perform well on All Range but poorly on Permuted Range because they were specifically designed for range queries. This shows that they exploit specific structure of the input workload and have limited adaptivity.

4. On these workloads, Laplace noise offers better error than Gaussian noise (for appropriately conservative settings of δ). This is because with Gaussian noise there is an additional $\approx \sqrt{\log(1/\delta)}$ term in the standard deviation of the noise, and this outweighs the benefit using the L_2 sensitivity norm instead of the L_1 sensitivity norm, despite the fact that we may be finding strategies that are further from optimal in the L_1 case.

Scalability We now demonstrate the scalability of OPT_0. Note that optimization time dominates in the low-dimensional setting, and the time for `measure` and `reconstruct` is small in comparison to that. The per-iteration time complexity only depends on the domain size, and not the contents of the workload. While the number

ϵ-differential privacy (Laplace noise)

Dataset	Workload	Identity	Laplace	DataCube	$\text{OPT}_{\otimes}$	OPT_{+}	OPT_{M}	HDMM	SVDB
CPH (5D)	SF1	23.20	70.71	-	7.30	30.55	9.56	7.30	-
	SF1+	32.50	141.42	-	10.23	42.31	12.88	10.23	-
CPS (5D)	All Marginals	5.38	45.25	18.49	4.85	4.85	4.84	4.84	2.63
	All Prefix-Marginals	98.06	56568.54	-	40.59	40.59	69.38	40.59	9.32
Adult (14D)	$\leq$ 3D Marginals	5352117.26	664.68	494.06	872.58	306.33	225.35	225.35	15.08
	2D Prefix-Marginals	475602516.60	138602.83	-	1119.16	484.07	553.56	484.07	-
Loans (12D)	Small Marginals	3330650.46	265.87	113.98	654.35	204.17	100.92	100.92	11.61
	Small Prefix-Marginals	15340082.96	11013.90	-	1707.67	485.67	288.29	288.29	-

Table 3.4: RMSE of HDMM strategies and baseline strategies on multi-dimensional workloads (ranging from 5D to 14D) for $\epsilon = 1.0$ with Laplace noise.

of iterations required for convergence may differ slightly based on the queries in the workload, for simplicity we measure the time required to run the optimization for 100 iterations on the All Range workload.

Figure 3.2 shows the amount of time required to run OPT_0 for various domain sizes. It shows that OPT_0 scales up to $n = 8192$, and runs for $n = 1024$ in under 10 seconds for Laplace noise and 1 minute for Gaussian noise. This difference occurs because the per-iteration time complexity is $O(pn^2)$ under Laplace noise but $O(n^3)$ under Gaussian noise. For $n = 8192$ it takes considerably longer, but is still feasible to run. We remark that trading a few hours of computation time for a meaningful reduction in error is typically a welcome trade-off in practice, especially since workloads can be optimized once and the resulting strategies reused many times. Additionally, we have a prototype implementation that uses GPUs and PyTorch, and we found that it is possible (although very time consuming) to scale up to $n = 16384$. Beyond this point, it quickly becomes infeasible to even represent the workload (or its Gram matrix) in matrix form, let alone optimize it.

3.10.2 Evaluating $\text{OPT}_{\otimes}$, OPT_{+}, and OPT_{M} on multi-dimensional workloads

We now shift our attention to the multi-dimensional setting.

(ϵ, δ)-differential privacy (Gaussian noise)

Dataset	Workload	Identity	Gaussian	DataCube	$\text{OPT}_{\otimes}$	OPT_{+}	OPT_{M}	HDMM	SVDB
CPH (5D)	SF1	69.33	29.87	-	9.80	75.66	14.31	9.80	-
	SF1+	97.08	42.25	-	10.90	84.16	15.88	10.90	-
CPS (5D)	All Marginals	16.08	23.90	19.53	7.85	7.85	7.86	7.85	7.85
	All Prefix-Marginals	292.93	844.94	-	29.48	29.49	104.09	29.48	27.85
Adult (14D)	$\leq$ 3D Marginals	15988375.02	91.59	77.36	82.42	899.04	46.44	46.44	45.06
	2D Prefix-Marginals	1420766966.19	1322.58	-	126.17	639.43	296.12	126.17	-
Loans (12D)	Small Marginals	9949649.11	57.92	37.37	81.51	631.40	34.91	34.91	34.67
	Small Prefix-Marginals	45825415.90	372.83	-	132.43	994.04	99.72	99.72	-

Table 3.5: RMSE of HDMM strategies and baseline strategies on multi-dimensional workloads (ranging from 5D to 14D) for $\epsilon = 1.0$ and $\delta = 10^{-6}$ with Gaussian noise.

Workloads We consider four multi-dimensional schemas and two workloads for each schema. The first two schemas are both census products, namely the Census of Population and Housing (CPH) and the Current Population Survey (CPS). These schemas each have 5 attributes each and domain sizes of about 1 million. The last two schemas, ADULT and LOANS are much higher-dimensional, having 15 and 12 attributes respectively.

For the CPH schema, we use two workloads, SF1 and SF1+ workloads, which include queries necessary to compute the statistics that appear in the Census' "Summary File 1" data releases at different geographic granularities. For the other schemas, we use workloads based on marginals and prefix-marginals. A prefix marginal is a query matrix of the form $\mathbb{Q} = \otimes_{i=1}^{d} \boldsymbol{Q}_i$ where $\boldsymbol{Q}_i \in \{\boldsymbol{T}, \boldsymbol{P}\}$ if i is a discretized numerical attribute, and $\boldsymbol{Q}_i =\in \{\boldsymbol{T}, \boldsymbol{I}\}$ otherwise, where $\boldsymbol{T}$, $\boldsymbol{I}$, and $\boldsymbol{P}$ are the "total", "identity", and "prefix" matrices, respectively. For CPS we use the workload of All Marginals and All Prefix-Marginals. For ADULT, we use All $\leq 3D$ MARGINALS and all 2D Prefix-Marginals. For LOANS, we use All Small Marginals and All Small Prefix-Marginals. A "Small" Marginal can be any k-way Marginal with less than 5000 cells. This means the workload will be an interesting combination of $0, 1, 2, \ldots, k$-way marginals.

We note that for the Adult and Loans schema, the domain is far too large to allow $\boldsymbol{p}$ to be represented in vector form. As a result, running HDMM as described in this paper would not be feasible. However, we remind the reader that in this section we are simply reporting *expected errors*, which we can compute efficiently without ever materializing $\boldsymbol{p}$.

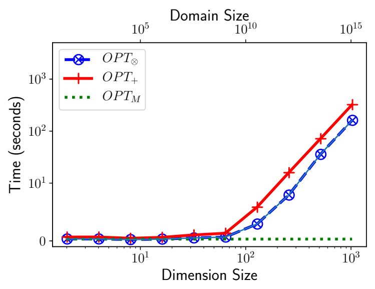
Mechanisms In the high-dimensional setting, there are far fewer data-independent mechanisms to choose from. We thus compare against Identity, Laplace, and Gaussian, which are the only methods from the previous section which are applicable and scalable to high-dimensional settings. In addition to these simple baselines, we also compare against DataCube, which is applicable in this setting, but only for (unweighted) marginal query workloads.

Results and Findings Table 3.4 and Table 3.5 report the RMSE of the baselines as well as each optimization operator. We compute the SVD bound when possible (i.e., the workload is either a single Kronecker product or a marginal query workload). We have four main findings which we enumerate below:

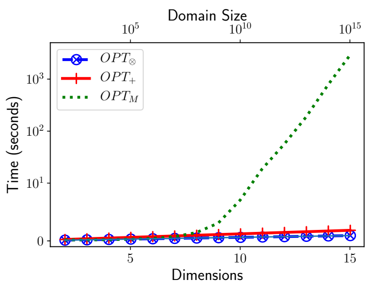
1. HDMM is better than all competitors on all tasks, and the magnitude of the improvement is as large as 38 for Laplace noise and 29 for Gaussian noise.

2. HDMM gets within a factor 1.06 of the SVD bound when it is possible to compute it for Gaussian noise. This is consistent with the theoretical result in Section 3.5 which justifies the defintion of $\mathsf{OPT}_{\otimes}$. For Laplace noise, the ratio is as high as 14, however.

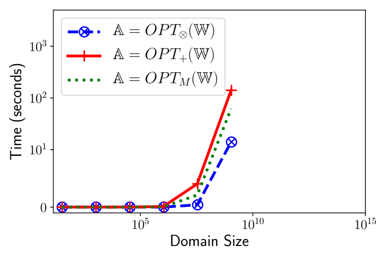
3. Gaussian noise offers lower error than Laplace noise for the two highest dimensional schemas, and comparable error for the two five-dimensional schemas. In contrast to the one-dimensional setting, this occurs because the savings from using the L_2 sensitivity norm outweighs the cost of $\approx \sqrt{\log(1/\delta)}$ to use Gaussian noise with (ϵ, δ)-differential privacy.



(a) Time to run `select` operators on 5-dimensional domains of size (c, c, c, c, c).



(b) Time to run `select` operators on d-dimensional domains of size $(10, \ldots, 10)$.



(c) Time to run `reconstruct` for each type of strategy with varying domain sizes.

Figure 3.3: Scalability of different components of HDMM when run on multi-dimensional domains of varying size and shape.

4. The parameterization that offers the lowest error differs based on the workload and the type of noise added. For example, OPT_M is always the best for workloads consisting of Marginals, but it is also sometimes the best for other workloads too. $\mathsf{OPT}_\otimes$ is the best for the CPH and CPS workloads, but not as good for the ADULT and LOANS workloads. OPT_+ is best for the low-dimensional Prefix Marginals workloads.

Scalability We now evaluate the scalability HDMM. The main factor that influences the scalability of HDMM is the domain. The optimization time primarily depends on the number of dimensions and the size of each dimension, while reconstruction time primarily depends on the total domain size. Thus, the bottleneck of HDMM depends on all of these factors in a nuanced way, and for some domains optimization will be the bottleneck, while for others reconstruction will be. We show how the key components scale with respect to these properties of the domain in Figure 3.3.

In Figure 3.3a, we fix the number of dimensions of the domain at $d = 5$ and vary the size of each dimension from $n_i = 2$ to $n_i = 1024$. We measure and report the optimization time for $\mathsf{OPT}_\otimes$, OPT_+, and OPT_M. We run $\mathsf{OPT}_\otimes$ for 100 inner iterations (in calls to OPT_0) and 5 outer iterations. We use a workload consisting of a union of 10

Kronecker products, where each subworkload is All Range queries. In Figure 3.3b, we fix the domain size of each dimension at $n_i = 10$ and vary the number of dimensions from $d = 2$ to $d = 15$. We again use the same workload as before. In Figure 3.3c, we use the strategies produced from Figure 3.3a, and measure the time required to perform the `reconstruct` step of HDMM.

From the figure we can see that the optimization time of $\mathsf{OPT}_{\otimes}$ and OPT_{+} primarily depends on the size of each dimension, rather than the number of dimensions. In contrast, the optimization time of $\mathsf{OPT}_{\mathsf{M}}$ primarily depends on the number of dimensions and not the size of each dimension. This confirms the theoretical complexity results. All three optimization operators are capable of running in settings where the total domain size is far too large to allow $\boldsymbol{p}$ to be represented in vector form. The figure also shows that we can solve the `reconstruct` step up to domains as large as 10^9. Beyond this point, it is infeasible to even represent $\boldsymbol{p}$ in vector form on the machine used for experiments.

3.11 Discussion and limitations

In this chapter, we presented HDMM, a general and scalable method for privately answering workloads of conjunctive linear queries. HDMM overcomes the main bottleneck of the Matrix Mechanism via implicit query matrix representations, and specialized optimized routines that exploit these implicit representations. In experiments, we ran HDMM on domains as large as $n = 10^9$. There are three main limitations to HDMM, which will motivate the work in Chapters 4 and 5.

First, HDMM represents the data vector explicitly, and while it scales very well with respect to the dimensionality of this object, it can be prohibitively expensive to instantiate the data vector explicitly for high-dimensional datasets which could have dozens of attributes and domain sizes much larger than 10^9. At the surface it is easy to misinterpret the name “high-dimensional” matrix mechanism, as HDMM is not actually able to run on datasets with more than a few attributes.

Second, like the matrix mechanism, HDMM solves an ordinary least squares problem to `reconstruct` the underlying data vector and estimate the workload query answers. While this produces unbiased estimates to the workload queries, it does not account for the non-negativity that we know holds in the true data vector. As a result, the reconstructed data vector could have negative counts, which is a consistency problem.

Third, HDMM provides unbiased answers to all workload queries. Mechanisms in this class are known to perform well in the big data/low privacy regime, but are often outperformed by biased mechanisms in the small data/high privacy regime [46]. When the workload is large, and there is a limited amount of data and privacy budget available, it can be better to privately answer a small subset of the workload (obtaining unbiased answers to those queries), and infer unanswered queries through post-processing. While these estimates will inevitably be biased, the reduced variance on the answered queries can be a worthwhile trade-off.

CHAPTER 4

PRIVATEPGM: ESTIMATING HIGH-DIMENSIONAL DATA DISTRIBUTIONS FROM NOISY MARGINALS

4.1 Motivation

Differentially private mechanisms are inherently random, as required by Definition 4. This randomness sometimes leads to inconsistencies in the private observations. Resolving these inconsistencies intelligently by post-processing can often improve the utility of the privacy mechanism [47]. A common approach is to estimate the underlying data vector from the noisy evidence [47, 57, 55], and then use this to give consistent estimates to the original queries. The estimated data distribution can also be used in place of the true data to answer new queries. This `reconstruct` step is a critical component of many mechanisms, as it improves utility at no cost to privacy.

Existing methods for solving this `reconstruct` problem typically rely on an explicit representation of the data vector, so they do not scale effectively to high-dimensional settings. There are a few notable exceptions [41, 110, 37], but they are tailored to specific settings and/or are based on simple heuristics and do not provide as much utility as the other methods [57, 55]. In this work we show that graphical models provide a natural solution to this problem that is general, principled, and scalable.

Contributions In this section, we present PrivatePGM, a general-purpose algorithm for post-processing the output of an arbitrary privacy mechanism defined over discrete data that estimates a data vector that maximizes the likelihood of the observed output. This estimate can be converted into synthetic tabular data and used in downstream tasks in place of the true data. PrivatePGM avoids materializing the data vector explicitly, and instead uses a compact factored representation. It is highly effective in high-dimensional settings, as long as all of the measurements are defined with respect to the low-dimensional marginals. Furthermore, because PrivatePGM is based on the principle of maximum likelihood, it consistently improves the utility of existing privacy mechanisms by extracting more useful information from the same randomized output. Moreover, PrivatePGM can be used to enable algorithms like HDMM to scale to higher-dimensional settings than what was previously possible.

Organization In Section 4.2, we setup the notation and state the main problem we consider. In Section 4.3, we present two algorithms for estimating the data vector (in factored form) from noisy observations of its marginals. In Section 4.4, we present algorithms for answering downstream queries efficiently by exploiting the factored form of the data vector. In Section 4.5, we show how PrivatePGM can be integrated into existing mechanisms. In Section 4.6, we empirically evaluate our approach, by integrating it with existing differentially private mechanisms, and evaluating scalability and utility.

4.2 Problem Setup

Suppose we ran an arbitrary (ϵ, δ)-DP mechanism $\mathcal{A}$ on a discrete dataset D and observed the output $\boldsymbol{y} \sim \mathcal{A}(D)$. The *observations* or *measurements* $\boldsymbol{y}$ reveal noisy high-level aggregate information about the underlying data. Our goal is to use this information to solve two related problems: (1) recover an estimate of the underlying dataset D, for the purposes of (2) estimating the answers to new queries about D. We can formulate the first problem as an optimization problem, as shown below:

Problem 2. *Let $L : \mathcal{D} \rightarrow \mathbb{R}$ be a loss function that measures how well a dataset D explains the randomized output $\boldsymbol{y}$. Our goal is to solve:*

$$\hat{D} \in \underset{D \in \mathcal{D}}{\arg\min}\, L(D)$$

The loss function can be any measure of well a dataset D "explains" the noisy observations $\boldsymbol{y}$. A natural choice that is universally applicable is the negative log-likelihood, that is $L(D) = -\log \Pr[\mathcal{A}(D) = \boldsymbol{y}]$, although other choices are certainly possible and may be preferable in specific settings. From $\hat{D}$, we can derive consistent estimates for measured queries, and also infer answers to unmeasured queries. Unfortunately, for high-dimensional domains, solving this problem in its full generality is hopelessly intractable, as $\mathcal{D}$ is an intractably large discrete set. We will therefore seek to solve the continuous relaxation of this problem instead, and optimize over the space of distributions $\boldsymbol{p}$:

Problem 3. *Let $L : \mathbb{R}^n \rightarrow \mathbb{R}$ be a loss function that measures how well a distribution $\boldsymbol{p}$ explains the randomized output $\boldsymbol{y}$. Our goal is to solve:*

$$\hat{\boldsymbol{p}} \in \underset{\boldsymbol{p} \in \mathcal{S}}{\arg\min}\, L(\boldsymbol{p})$$

Above, S is the set of all possible (normalized) data vectors, or non-negative vectors that sum to 1. In this chapter, we will assume that the number of records N is known, so we can easily convert back and forth between the normalized and unnormalized data vector representations. If N is not known, we can usually estimate it from the noisy observations [73]. Problem 3 is a simple generalization of prior problem formulations [47, 57, 80], which have primarily focused on the setting where $\mathcal{A}$ answers linear queries, and the loss function simplifies to $L(\boldsymbol{p}) = \|\boldsymbol{Q}\boldsymbol{p} - \boldsymbol{y}\|$ for some matrix $\boldsymbol{Q}$. If n is sufficiently small, this is a simple problem to solve using gradient-based techniques. However, in high-dimensional settings, we often have $n \gg 10^9$, and as a result $\boldsymbol{p}$ is far too large to work with directly.

To make this problem tractable, we will require that the mechanism $\mathcal{A}$ and therefore the loss function L only depends on D through some collection of its low-dimensional marginals $\boldsymbol{\mu}_r = M_r(D)$ for $r \in \mathcal{C}$. Let $\boldsymbol{\mu} = (\boldsymbol{\mu}_r)_{r \in \mathcal{C}}$ denote the *concatenated vector of marginals*, or simply the *clique marginals*. Many published mechanisms for high-dimensional data satisfy this assumption, so the requirement is not overly restrictive [110, 41, 69, 37, 84, 16, 116, 70, 71, 20, 8]. Moreover, by setting $\mathcal{C} = \{r\}$ where $r = [d]$, the mechanism can depend on the entire data vector $\boldsymbol{p}$, so there is no loss in generality in this problem formulation. However, when the set $\mathcal{C}$ contains lower-dimensional marginals, we wish to exploit this fact to solve this problem more efficiently.

To facilitate this problem formulation, it is helpful to introduce the function $M_{\mathcal{C}} : \mathcal{D} \rightarrow \mathbb{R}^k$ where $k = \sum_{r \in \mathcal{C}} n_r$, which computes the concatenated vector of marginals for a given set of cliques $\mathcal{C}$, i.e., $\boldsymbol{\mu} = M_{\mathcal{C}}(D) = (M_r(D))_{r \in \mathcal{C}} = (\boldsymbol{\mu}_r)_{r \in \mathcal{C}}$. We will also let $\boldsymbol{M}_{\mathcal{C}}$ denote the corresponding linear transformation, i.e., $\boldsymbol{\mu} = \boldsymbol{M}_{\mathcal{C}} \boldsymbol{p}$. To make the problem setting more concrete, it is useful to consider the canonical mechanism defined by $\mathcal{A}(D) = \boldsymbol{Q} M_{\mathcal{C}}(D) + \boldsymbol{\xi}$ where $\mathcal{C}$ is a set of cliques, $\boldsymbol{Q}$ is a query matrix, and $\boldsymbol{\xi}$ is a vector of zero-centered noise. A natural loss function for this mechanism is $L(\boldsymbol{p}) = \|\boldsymbol{Q}\boldsymbol{M}_{\mathcal{C}}\boldsymbol{p} - \boldsymbol{y}\|_2^2$. Indeed, this loss function is proportional to the negative log likelihood when $\boldsymbol{\xi}$ is normally distributed, although the loss function is a natural choice for other zero-centered noise distributions as well. While the techniques we present in this paper apply to more general mechanisms, the canonical mechanism above is helpful to gain more intuition about the problem setup and applications.

4.3 Estimating the data distribution

We take a two-step approach to estimating $\boldsymbol{p}$. In the first step, we solve for the best-fitting $\boldsymbol{\mu}$ by solving the optimization problem stated below. In the second step, we identify a distribution $\boldsymbol{p}_{\boldsymbol{\theta}}$ that has marginals $\boldsymbol{\mu}$.

Problem 4 (Marginal-based Optimization)**.** *Given a loss function $L(\boldsymbol{\mu})$, find*

$$\hat{\boldsymbol{\mu}} \in \operatorname*{arg\,min}_{\boldsymbol{\mu} \in \mathcal{M}(\mathcal{C})} L(\boldsymbol{\mu})$$

where $\mathcal{M}(\mathcal{C}) = \{\boldsymbol{M}_{\mathcal{C}}\boldsymbol{p} \mid \boldsymbol{p} \in S\}$ is the marginal polytope, or the set of vectors that are the marginals of some data vector.

The main benefit of this problem reformulation is that the marginal vector $\boldsymbol{\mu}$ will often be much smaller than $\boldsymbol{p}$, so the new problem can be more efficiently solved. However, there are three main technical challenges to overcome. First, Problem 4 is an optimization problem over the marginal polytope, which is nontrivial to solve because the constraint set has complex combinatorial structure. Second, the solution to Problem 4 only gives us $\hat{\boldsymbol{\mu}}$, but we are ultimately interested in the full joint distribution $\hat{\boldsymbol{p}}$, and there are infinitely many data vectors $\boldsymbol{p}$ that are compatible with a given marginal vector $\boldsymbol{\mu}$. Third, in the high-dimensional setting we cannot afford to represent $\boldsymbol{p}$ explicitly, as $n \gg 10^9$. To overcome these challenges, we will use undirected graphical models. A graphical model provides a compact factored representation of a joint distribution, and is defined formally below:

Definition 25 (Graphical model)**.** *Let*

$$\boldsymbol{p}_{\boldsymbol{\theta}}(x) = \frac{1}{Z} \exp\Big(\sum_{r \in \mathcal{C}} \boldsymbol{\theta}_r(x_r)\Big)$$

where Z is a normalization constant and $\boldsymbol{\theta}_r \in \mathbb{R}^{n_r}$. This distribution is a graphical model that factors over a collection of cliques $\mathcal{C}$. The real numbers $\boldsymbol{\theta}_r(x_r)$ are known as the log-potentials, or simply the parameters of the model.

The distribution $\boldsymbol{p}_\theta$ is completely defined by the parameter vector $\boldsymbol{\theta} = (\boldsymbol{\theta}_r)_{r\in\mathcal{C}}$, which matches the marginal vector $\boldsymbol{\mu}$ in size and indexing. The relationship between these two vectors is central to graphical models [97]:

- A parameter vector $\boldsymbol{\theta}$ determines a unique marginal vector $\boldsymbol{\mu}_{\boldsymbol{\theta}} \in \mathcal{M}(\mathcal{C})$, defined as $\boldsymbol{\mu}_{\boldsymbol{\theta}} = \boldsymbol{M}_{\mathcal{C}}\boldsymbol{p}_{\boldsymbol{\theta}}$, the marginals of $\boldsymbol{p}_{\boldsymbol{\theta}}$. *Marginal inference* is the problem of (efficiently) computing $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ from $\boldsymbol{\theta}$. It can be solved exactly by algorithms such as *variable elimination* or *belief propagation* with a junction tree [53]. We denote by `MARGINAL-ORACLE` an algorithm that outputs $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ on input $\boldsymbol{\theta}$.

- For every $\boldsymbol{\mu} \in \mathcal{M}(\mathcal{C})$ with positive entries, there is a unique distribution $\boldsymbol{p}_{\boldsymbol{\theta}}$ in the family of graphical models with cliques $\mathcal{C}$ that has marginals $\boldsymbol{\mu}$. Moreover, $\boldsymbol{p}_{\boldsymbol{\theta}}$ has *maximum entropy* among all distributions with marginals $\boldsymbol{\mu}$.

4.3.1 Solving Problem 4 with proximal algorithms

Now that we have outlined out high-level approach, we are now ready to present two algorithms for actually finding $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\theta}}$. Both are proximal algorithms for solving convex problems with "simple" constraints [81]. The first and simpler of the two is shown in Algorithm 2. It is an T-step procedure that finds both the marginals $\boldsymbol{\mu}$, as well as the parameter vector $\boldsymbol{\theta}$ of the corresponding graphical model. If run until convergence, the returned vector $\boldsymbol{\mu}$ solves Problem 4, and $\boldsymbol{p}_{\boldsymbol{\theta}}$ solves Problem 3. Each step of the procedure invokes `MARGINAL-ORACLE`, which is a black-box algorithm for computing the clique marginals $\boldsymbol{\mu}$ of the graphical model from the parameters $\boldsymbol{\theta}$. This is the problem of *marginal inference* in a graphical model. `MARGINAL-ORACLE` may be any marginal inference routine — we use belief propagation on a junction tree, which is the standard algorithm used for this problem [53]. As we will discuss below, this algorithm is an instance of mirror descent, and hence inherits its convergence guarantees. It will converge for any convex loss function L at a $O(1/\sqrt{t})$ rate,[1] even ones that are not smooth, such as the L_1 loss.

The complexity of Algorithm 2 is closely tied to the complexity of `MARGINAL-ORACLE`, or belief propagation in the underlying graphical model. Belief propagation exploits the factored representation of $\boldsymbol{p}_{\boldsymbol{\theta}}$ to calculate the marginals $\boldsymbol{\mu}$ more efficiently than the naïve approach which simply materializes $\boldsymbol{p}_{\boldsymbol{\theta}}$ explicitly as a vector, then computes

[1]That is, $L(\boldsymbol{\mu}^t) - L(\boldsymbol{\mu}^*) \in O(1/\sqrt{t})$.

Algorithm 2 Proximal Estimation Algorithm

```
Input: Convex loss function $L(\boldsymbol{\mu})$
Output: Estimated marginals $\boldsymbol{\mu}$ and parameters $\boldsymbol{\theta}$
$\boldsymbol{\theta} = \mathbf{0}$
for $t = 1, \dots, T$ do
    $\boldsymbol{\mu} = \texttt{MARGINAL-ORACLE}(\boldsymbol{\theta})$
    $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta_t \nabla L(\boldsymbol{\mu})$
end for
return $\boldsymbol{\mu}, \boldsymbol{\theta}$
```

$\boldsymbol{\mu} = \boldsymbol{M}_{\mathcal{C}} \boldsymbol{p}_{\boldsymbol{\theta}}$. In fact, belief propagation avoids enumerating the entries of $\boldsymbol{p}_{\boldsymbol{\theta}}$ one-by-one all-together. Instead, all computations are done directly in terms of the parameters $\boldsymbol{\theta}$ and intermediate objects called "messages". The complexity of belief propagation does not depend directly on n, and therefore it can handle our target case $n \gg 10^9$. The main factor that influences the complexity of belief propagation is the *structure* of the underlying graphical model. For "nice" graphical models, belief propagation is exponentially faster than the naïve approach, and is tractable in high-dimensional settings. In the worst case, however, belief propagation is no better than the naïve approach. We study the scalability of Algorithm 2 in Section 4.6.

4.3.2 Derivation of the update equations

Algorithm 2 is inspired by the entropic mirror descent algorithm for solving convex optimization problems over the probability simplex [7]. The iterates of the optimization are obtained by solving simpler optimization problems of the form:

$$\boldsymbol{\mu}^{t+1} = \operatorname*{arg\,min}_{\boldsymbol{\mu} \in \mathcal{M}(\mathcal{C})} \boldsymbol{\mu}^\top \nabla L(\boldsymbol{\mu}^t) + \frac{1}{\eta_t} D(\boldsymbol{\mu}, \boldsymbol{\mu}^t) \tag{4.1}$$

where D is a Bregman divergence, which has the form $D(\boldsymbol{\mu}, \boldsymbol{\mu}^t) = \psi(\boldsymbol{\mu}) - \psi(\boldsymbol{\mu}^t) - (\boldsymbol{\mu} - \boldsymbol{\mu}^t)^\top \nabla \psi(\boldsymbol{\mu}^t)$ for a strongly convex and continuously differentiable function ψ. ψ should be chosen to reflect the geometry of the constraint set, so that the subproblem above can be solved efficiently. Here we use $\psi = -H$, the negative Shannon entropy of the graphical model $\boldsymbol{p}_{\boldsymbol{\theta}}$ with marginals $\boldsymbol{\mu}$. This choice was inspired by Vilnis et al., who considered a similar optimization problem [96]. Since we assumed above that $\boldsymbol{\mu}$ are marginals of the cliques of a junction tree, the Shannon entropy is convex and easily computed as a function of $\boldsymbol{\mu}$ alone [97]. We show below that the subproblem can be efficiently solved with a call to `MARGINAL-ORACLE`.

$$\begin{aligned}
\boldsymbol{\mu}^{t+1} &= \operatorname*{arg\,min}_{\boldsymbol{\mu} \in \mathcal{M}(\mathcal{C})} \boldsymbol{\mu}^\top \nabla L(\boldsymbol{\mu}^t) + \frac{1}{\eta_t} D(\boldsymbol{\mu}, \boldsymbol{\mu}^t) \\
&= \operatorname*{arg\,min}_{\boldsymbol{\mu} \in \mathcal{M}(\mathcal{C})} \boldsymbol{\mu}^\top \nabla L(\boldsymbol{\mu}^t) + \frac{1}{\eta_t} \Big(- H(\boldsymbol{\mu}) + \boldsymbol{\mu}^\top \nabla H(\boldsymbol{\mu}^t) \Big) && \text{(substitution)} \\
&= \operatorname*{arg\,min}_{\boldsymbol{\mu} \in \mathcal{M}(\mathcal{C})} \boldsymbol{\mu}^\top \Big(\eta_t \nabla L(\boldsymbol{\mu}^t) + \nabla H(\boldsymbol{\mu}^t) \Big) - H(\boldsymbol{\mu}) && \text{(algebraic manipulation)} \\
&= \operatorname*{arg\,min}_{\boldsymbol{\mu} \in \mathcal{M}(\mathcal{C})} \boldsymbol{\mu}^\top \Big(\eta_t \nabla L(\boldsymbol{\mu}^t) - \boldsymbol{\theta}^t \Big) - H(\boldsymbol{\mu}) && (\nabla H(\boldsymbol{\mu}^t) = -\boldsymbol{\theta}^t \text{ [97]}) \\
&= \texttt{MARGINAL-ORACLE}\big(\boldsymbol{\theta}^t - \eta_t \nabla L(\boldsymbol{\mu}^t) \big) && \text{(inference} \leftrightarrow \text{energy minimization [97])}
\end{aligned}$$

The first three steps are simple algebraic manipulation of the mirror descent update equation. The final two steps use the observation that $\nabla H(\boldsymbol{\mu}^t) = -\boldsymbol{\theta}^t$ and that marginal inference can be cast as the following optimization problem: [97, 96] $\texttt{MARGINAL-ORACLE}(\boldsymbol{\theta}) = \operatorname{arg\,min}_{\boldsymbol{\mu} \in \mathcal{M}(\mathcal{C})} -\boldsymbol{\mu}^\top \boldsymbol{\theta} - H(\boldsymbol{\mu})$.

4.3.3 Accelerated proximal algorithm

We now present a related algorithm which is based on the same principles as Algorithm 2 but has an improved $O(1/t^2)$ convergence rate for convex loss functions with Lipschitz continuous gradients. Algorithm 3 is based on Nesterov's accelerated dual averaging approach [78, 99, 96]. The per-iteration complexity is the same as Algorithm 2 as it requires calling the `MARGINAL-ORACLE` once, but this algorithm will converge in fewer iterations. Algorithm 3 has the advantage of not requiring a step size to be set, but it requires knowledge of the Lipschitz constant of ∇L. For the standard L_2 loss with linear measurements, this is equal to the largest eigenvalue of $\boldsymbol{Q}^\top \boldsymbol{Q}$. The derivation of this algorithm is similar to the derivation of Algorithm 2.

Algorithm 3 Accelerated Proximal Estimation Algorithm

```
Input: Loss function $L(\boldsymbol{\mu})$ between $\boldsymbol{\mu}$ and $\boldsymbol{y}$
Output: Estimated data distribution $\hat{\boldsymbol{p}}_{\boldsymbol{\theta}}$
$K$ = Lipschitz constant of $\nabla L$
$\bar{\boldsymbol{g}} = \mathbf{0}$
$\boldsymbol{\nu}, \boldsymbol{\mu} = \texttt{MARGINAL-ORACLE}(\mathbf{0})$
for $t = 1, \dots, T$ do
    $c = \frac{2}{t+1}$
    $\boldsymbol{\omega} = (1-c)\boldsymbol{\mu} + c\boldsymbol{\nu}$
    $\bar{\boldsymbol{g}} = (1-c)\bar{\boldsymbol{g}} + c\nabla L(\boldsymbol{\omega})$
    $\boldsymbol{\theta} = \frac{-t(t+1)}{4K}\bar{\boldsymbol{g}}$
    $\boldsymbol{\nu} = \texttt{MARGINAL-ORACLE}(\boldsymbol{\theta})$
    $\boldsymbol{\mu} = (1-c)\boldsymbol{\mu} + c\boldsymbol{\nu}$
end for
return graphical model $\hat{\boldsymbol{p}}_{\boldsymbol{\theta}}$ with marginals $\boldsymbol{\mu}$
```

Remark 1. *The complexity of Algorithms 2 and 3 is closely related to the complexity of MARGINAL-ORACLE, which in turn depends on the structure of the underlying graphical model, which in turn depends on the cliques that were measured by the mechanism. When these cliques form a tree, MARGINAL-ORACLE and PrivatePGM are highly scalable. In general, the complexity of these techniques depends on a quantity known as the junction tree size. While it requires some expertise in graphical models to understand these nuances, we can easily compute this quantity as a function of the measured cliques using a function JT-SIZE$(r_1, \dots, r_t)$. In this chapter, we only consider cliques which lead to tractable graphical models, and implicitly assume JT-SIZE is sufficiently small. In Chapter 5, we will use this function explicitly to ensure the cliques we measure lead to tractable models.*

4.4 Estimating new query answers

Now that we have shown how to estimate $\boldsymbol{p}_{\boldsymbol{\theta}}$, we will show how to use it to efficiently estimate the answers to new queries, without materializing $\boldsymbol{p}_{\boldsymbol{\theta}}$ explicitly. We will describe two approaches that can compute linear transformations of marginals.

Algorithm 4 Inference for conjunctive linear queries

```
Input: Parameters $\boldsymbol{\theta}$, query matrix $\mathbb{Q} = \boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d$
Output: Query answers $\mathbb{Q}\,\boldsymbol{p}_{\boldsymbol{\theta}}$
$\psi = \{\exp(\boldsymbol{\theta}_r) \mid r \in \mathcal{C}\} \cup \{\boldsymbol{Q}_i \mid i \in [d]\}$
$Z = \texttt{NORMALIZATION-CONSTANT}(\boldsymbol{\theta})$
return $\texttt{VARIABLE-ELIM}(\psi, \Omega)/Z$
```

Additionally, we will show that we can generate synthetic tabular data from $\boldsymbol{p}_{\boldsymbol{\theta}}$ which we can use to estimate answers to virtually any new query.

4.4.1 Marginal queries

Given a clique r we would like to compute $\boldsymbol{\mu}_r = \boldsymbol{M}_r \boldsymbol{p}_{\boldsymbol{\theta}}$, which will serve as an estimate for $M_r(D)$. However, we cannot afford to compute $\boldsymbol{\mu}_r$ directly using the formula above. If $r \in \mathcal{C}_+$, where $\mathcal{C}_+$ is the downward closure of $\mathcal{C}$, then we can immediately calculate $\boldsymbol{\mu}_r$ from the concatenated vector of marginals $\boldsymbol{\mu}$ returned by Algorithm 2. Alternatively, if $r \notin \mathcal{C}_+$, then we instead use the variable elimination algorithm [53], which is similar in spirit to belief propagation for computing $\boldsymbol{\mu}$, but is able to compute marginals that are not in the $\mathcal{C}$.

4.4.2 Conjunctive linear queries

Algorithm 4 gives an efficient algorithm for answering *conjunctive linear queries*, represented as a Kronecker product $\mathbb{Q} = \boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d$ (Definition 18). It can be understood as follows: for a particular query index $z = (z_1, \dots, z_d)$, write $f(z, x) = \mathbb{Q}(z, x)\boldsymbol{p}_{\boldsymbol{\theta}}(x) = \boldsymbol{p}_{\boldsymbol{\theta}}(x) \prod_i \boldsymbol{Q}_i(z_i, x_i)$. This can be viewed as an augmented graphical model on the variables z and x where we have introduced new pairwise factors between each (x_i, z_i) pair defined by the query matrix $\boldsymbol{Q}_i$. Unlike a regular graphical model, the new factors can contain negative values. The query answers are obtained by multiplying $\mathbb{Q}$ and $\boldsymbol{p}$, which sums over x. The z^{th} answer is given by:

$$\begin{aligned}(\mathbb{Q}\boldsymbol{p}_{\boldsymbol{\theta}})(z) &= \sum_{x \in \Omega} \mathbb{Q}(z, x)\boldsymbol{p}_{\boldsymbol{\theta}}(x) \\ &= \frac{1}{Z}\sum_{x \in \Omega} \prod_{i=1}^{d} \boldsymbol{Q}_i(z_i, x_i) \prod_{r \in \mathcal{C}} \exp[\boldsymbol{\theta}_r(x_r)]\end{aligned}$$

This can be understood as marginalizing over the x variables in the augmented model $f(z, x)$, leaving us with only the variables z. The resulting marginal in the augmented graphical model contains the desired query answers. The `VARIABLE-ELIM` routine referenced in the algorithm is standard variable elimination to perform this marginalization; it can handle negative values with no modification. The term Z is the normalization constant, which can be computed using an algorithm similar to `MARGINAL-ORACLE`.

4.4.3 Synthetic data

One approach to estimate answers to more exotic queries is to simply generate synthetic data $\hat{D}$ from $\hat{\boldsymbol{p}}_{\boldsymbol{\theta}}$. This synthetic dataset $\hat{D}$ can then be used as a drop-in replacement for D to estimate the answer to any query. A naïve method to generate synthetic data is to simply sample from $\hat{\boldsymbol{p}}_{\boldsymbol{\theta}}$. It is well-known how to efficiently sample from a graphical model [97], and the complexity of sampling is roughly comparable to that of `MARGINAL-ORACLE`. The basic idea is to iterate through the attributes $1, \dots, d$ in a carefully chosen order, and sample data for one attribute at a time, conditioned on the data generated during previous iterations. This major drawback of this approach is the introduction of additional randomness due to sampling, which introduces error to $\hat{D}$ that was not present in $\hat{\boldsymbol{p}}_{\boldsymbol{\theta}}$. We thus propose a different procedure that replaces the sampling step with a *randomized rounding* step [70]. While this new procedure still has some randomness, it has far smaller variance than the sampling scheme. Some deviation between $\hat{D}$ and $\hat{\boldsymbol{p}}_{\boldsymbol{\theta}}$ is inevitable as $\hat{D}$ is a discrete approximation of $\hat{\boldsymbol{p}}$.

4.5 Integrating PrivatePGM into existing mechanisms

Next we describe how PrivatePGM can improve the accuracy and/or scalability of four state-of-the-art mechanisms: MWEM, PrivBayes, HDMM, and DualQuery. Table 4.1 provides a high-level summary of the impact of PrivatePGM on the accuracy and scalability of four mechanisms. More details on how PrivatePGM is integrated into these mechanisms are provided in the paragraphs below.

Table 4.1: Breakdown of how PrivatePGM improves the accuracy and scalability of existing mechanisms.

Mechanism	Accuracy	Scalability
MWEM [73]		✓
PrivBayes [110]	✓	
HDMM [73]	✓	✓
DualQuery [37]	✓	

MWEM The multiplicative weights exponential mechanism [41] is an active-learning style algorithm that is designed to answer a workload of linear queries. MWEM maintains an approximation of the data distribution and at each time step selects the worst approximated query from the workload via the exponential mechanism [75]. It then measures the query using the Laplace mechanism and then updates the approximate data distribution by incorporating the measured information using the multiplicative weights update rule.

It is infeasible to represent $\boldsymbol{p}$ explicitly for high-dimensional data, so this version of MWEM is only applicable to relatively low-dimensional data. Hardt et al. describe an enhanced version of MWEM, which we call *factored* MWEM, that is able to avoid materializing this vector explicitly, in the special case when the measured queries decompose over disjoint subsets of attributes. In that case, $\boldsymbol{p}$ is represented implicitly as a product of independent distributions over smaller domains, i.e., $\boldsymbol{p}(x) = \prod_{r \in \mathcal{C}} \boldsymbol{\mu}_r(x_r)$, and the update is done on one group at a time. However, this enhancement breaks down for measurements on overlapping subsets of attributes in high-dimensional data, so MWEM is still generally infeasible to run except on simple workloads.

We can construct a new algorithm, MWEM+PGM, by incorporating PrivatePGM to this this procedure. Specifically, we can replace the multiplicative weights update step with a call to Algorithm 2 using the standard L_2 loss function (on all measurements up to that point in the algorithm). By doing so, we learn a compact graphical model representation of $\hat{\boldsymbol{p}}$, which avoids materializing the full $\boldsymbol{p}$ vector even when the measured queries overlap in complicated ways. MWEM+PGM scales better than factored MWEM and runs in settings where it was previously infeasible. Interestingly, in settings where both MWEM and MWEM+PGM run, they produce identical results, even though they appear to use different procedures for estimating $\boldsymbol{p}$ on the surface. Upon closer inspection, this can be explained by the fact that the MWEM update is closely related to the entropic mirror descent update [7], and, if iterated until convergence (as is done in practice), solves the same L_2 minimization problem that we use[73].

PrivBayes PrivBayes [110] is a differentially private mechanism that generates synthetic data. It first spends half the privacy budget to learn a Bayesian network structure that captures the dependencies in the data, and then uses the remaining privacy budget to measure the statistics—which are marginals—necessary to learn the Bayesian network parameters. PrivBayes uses a heuristic of truncating negative entries of the noisy marginals and normalizing to get conditional probability tables. It then samples a synthetic dataset of N records from the Bayesian network from which consistent answers to workload queries can be derived. While this is simple and efficient, the heuristic does not properly account for measurement noise and sampling may introduce unnecessary error.

We can incorporate PrivatePGM into PrivBayes by replacing it's model estimation step with a call to Algorithm 2 to create the new mechanism PrivBayes+PGM.Then we can answer new queries directly with the learned model $\boldsymbol{p}_{\boldsymbol{\theta}}$, or generate synthetic data as PrivBayes does.

HDMM The high-dimensional matrix mechanism [69] was described in detail in Chapter 3. The main bottleneck of HDMM is representing $\boldsymbol{p}$, which it must estimate during the `reconstruct` step via least squares problem: $\hat{\boldsymbol{p}} = \arg\min_{\boldsymbol{p}} \|\mathbb{Q}\boldsymbol{p} - \boldsymbol{y}\|_2$. We can replace the HDMM estimation procedure with Algorithm 2, using the same L_2 loss function to create HDMM+PGM. If the workload contains queries over low-dimensional marginals, then the strategy $\mathbb{Q}$ will also contain queries over the low-dimensional marginals (when using OPT_+ and OPT_M). In this case, the main assumption of PrivatePGM is satisfied, and therefore we can expect HDMM+PGM to scale much better than HDMM by itself. In addition to improving scalability of HDMM, we expect HDMM+PGM to provide better accuracy as well, since Algorithm 2 considers the non-negativity constraint on $\boldsymbol{p}$, which HDMM does not.

DualQuery DualQuery [37] is an iterative algorithm inspired by the same two-player game underlying MWEM. It generates synthetic data to approximate the true data on a workload of linear queries. DualQuery maintains a distribution over the workload queries that depends on the true data so that poorly approximated queries have higher probability mass. In each iteration, samples are drawn from the query distribution, which are proven to be differentially private. The sampled queries are then used to find a single record from the data domain (without accessing the protected data), which is added to the synthetic database.

The measurements — i.e., the random outcomes from the privacy mechanism — are the queries sampled in each iteration. Even though these are very different from the canonical case of linear measurements we used to motivate PrivatePGM, we can still express the log-likelihood as a function of $\boldsymbol{p}$ and select $\boldsymbol{p}$ to maximize the log-likelihood using Algorithm 2. Because the log-likelihood only depends on $\boldsymbol{p}$

through the answers to the workload queries, if the workload can be expressed in terms of $\boldsymbol{\mu}$ instead, the log-likelihood can as well. Thus, after running DualQuery, we can call Algorithm 2 with this custom loss function to estimate the data distribution, creating the new mechanism DualQuery+PGM. We can then use the estimated model $\boldsymbol{p_\theta}$ to estimate downstream queries directly, or to generate synthetic data as DualQuery does. Additional details on this approach are given in the supplementary material.

New Mechanisms We have demonstrated above that PrivatePGM can be integrated into a variety of existing mechanisms for discrete data. However, we developed PrivatePGM not just to improve existing mechanisms, but to act as a general-purpose tool that can simplify the design of *future* mechanisms. For mechanisms in the `select-measure-reconstruct` paradigm, PrivatePGM provides a principled and scalable solution to the `reconstruct` step, which in turn allows the mechanism designer to focus on the orthogonal `select` step. Indeed, in Chapter 5, we develop a new mechanism for differentially private synthetic data generation that utilizes PrivatePGM in this way.

4.6 Experimental evaluation

In this section, we measure the accuracy and scalability improvements enabled by PrivatePGM when it is incorporated into existing privacy mechanisms.

4.6.1 Adding PrivatePGM to existing algorithms

To demonstrate the usefulness of our technology, we run four privacy mechanisms (MWEM, PrivBayes, HDMM, DualQuery) with and without PrivatePGM. These mechanisms are run with a privacy budget of $\epsilon = 1.0$ (and $\delta = 0.001$ for DualQuery). We repeat each experiment five times and report the median workload error.

We use four datasets in our experiments, summarized in Table 5.2. Each dataset consists of a collection of categorical and numerical attributes (with the latter discretized into 100 bins).

Dataset	Records	Attributes	Domain	Queries
TITANIC	1304	9	3×10^8	4851
ADULT	48842	15	1×10^{19}	62876
LOANS	42535	48	5×10^{80}	362201
STROKE	19434	110	4×10^{104}	17716

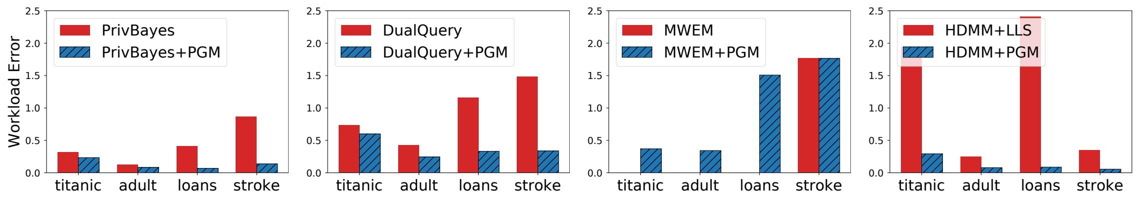
Table 4.2: Datasets used in experiments along with the number of queries in the workload used with the dataset.

We evaluate error with respect to a workload of fifteen randomly chosen three way range-marginals. Specifically, for each attribute $i \in [d]$, we define $\boldsymbol{W}_i = \boldsymbol{I}$ (the identity matrix) if i is a categorical attribute, and $\boldsymbol{W}_i = \boldsymbol{P}$ (the prefix matrix) if i is a discretized numeric attribute. Then, we sample 15 size three cliques, and for each clique $r = (i_1, i_2, i_3)$, we construct the query matrix $\mathbb{W}_r = \boldsymbol{W}_{i_1} \otimes \boldsymbol{W}_{i_2} \otimes \boldsymbol{W}_{i_3}$. Error is measured by a normalized L_1 distance between true and estimated answers to the workload queries.

$$Error(\boldsymbol{\mu}, \hat{\boldsymbol{\mu}}) = \frac{1}{|\mathcal{C}|} \sum_{r \in \mathcal{C}} \frac{\|\mathbb{W}_r \boldsymbol{\mu}_r - \mathbb{W}_r \hat{\boldsymbol{\mu}}_r\|_1}{2 \|\mathbb{W}_r \boldsymbol{\mu}_r\|_1}$$

where $\boldsymbol{\mu}$ and $\hat{\boldsymbol{\mu}}$ are the true and estimated data marginals, respectively. The summand is related to the total variation distance, and is equal in the special case when $\mathbb{W}_r = \mathbb{I}$.

Improved accuracy. PrivBayes and DualQuery are highly scalable algorithms supporting the large domains considered here. Figures 4.1a and 4.1b show that incorporating PrivatePGM significantly improves their accuracy. For PrivBayes, workload error is reduced by a factor of 6× and 7× on the LOANS and STROKE datasets, respectively, and 30% for ADULT. For DualQuery, we also observe very significant error reductions of 1.2×, 1.8×, 3.5×, and 4.4×.



(a) PrivBayes (b) DualQuery (c) MWEM (d) HDMM

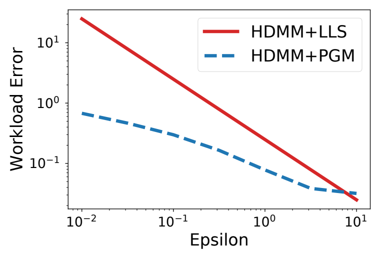
Figure 4.1: Workload error of four mechanisms on four datasets, with and without PrivatePGM for $\epsilon = 1.0$.

Replacing infeasible estimation methods. The MWEM and HDMM algorithms fail to run on the datasets and workloads we consider because both require representations of the data vector that are too large to maintain in memory. However, incorporating PrivatePGM makes these algorithms feasible in these settings.

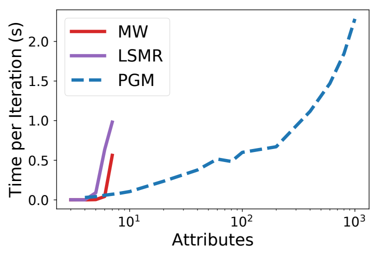
As Figure 4.1c shows, for the first three datasets, MWEM crashed before completing because it ran out of memory or timed out. It was able to run successfully on the 105-dimensional STROKE dataset, because we are using a *factored* MWEM implementation, and the cliques in the workload did not overlap too much.

HDMM fails to run on all datasets, so for the purpose of comparison, we run a modified version of the algorithm (denoted HDMM+LLS) which uses local least squares independently over each measurement marginal instead of global least squares over the full data vector. While scalable, Figure 4.1d shows that this heuristic is substantially worse than PrivatePGM, especially on the TITANIC and LOANS dataset. Incorporating PrivatePGM offers error reductions of 6.6×, 3.2×, 27×, and 6.3×. These improvements primarily stem from non-negativity and global consistency offered by PrivatePGM.

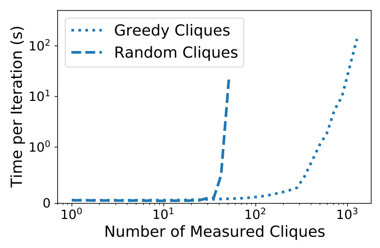
Varying epsilon. While ϵ is set to 1 in Figure 4.1, in Figure 4.2a we look at the impact of varying ϵ, for a fixed dataset and measurement set. We use the ADULT dataset and the measurements selected by HDMM, (which do not depend on ϵ). The magnitude of the improvement offered by PrivatePGM increases as ϵ decreases. At $\epsilon = 0.3$ and below, the mechanism has virtually no utility without PrivatePGM. At the



(a)



(b)



(c)

Figure 4.2: (a) Error of HDMM variants on ADULT as a function of ϵ. (b) Scalability of estimation algorithms. (c) Scalability of PrivatePGM for randomly and greedily selected cliques.

highest ϵ of 10.0, HDMM+LLS actually offers slightly lower error than HDMM+PGM on the workload, although both have very low error in an absolute sense. The error of HDMM+PGM on the *measurements* is still better by more than a factor of three at this privacy level. This behavior has been observed before in the low-dimensional setting, where the ordinary least squares estimator generalizes better than the non-negative least squares estimator for workloads with range queries [60].

4.6.2 The scalability of PrivatePGM

We now evaluate the scalability of our approach compared with two other general-purpose estimation techniques: multiplicative weights (MW) [41] and iterative ordinary least squares (LSMR) [34, 109]. We omit from comparison PrivBayes estimation and DualQuery estimation because they are special-purpose estimation methods that cannot handle arbitrary linear measurements. We conduct two separate scalability experiments, to demonstrate how different factors influence the scalability of PrivatePGM.

Varying number of attributes We use synthetic data so that we can systematically vary the domain size and the number of attributes. We consider the simple Laplace mechanism that adds noise directly to the three way marginals on attributes $r = (i, i+1, i+2)$ for $1 \leq i \leq d-2$. In Figure 4.2b, we vary the number of attributes from 3 to 1000, fixing $|\Omega_i| = 10$ for each attribute i, and plot the time per iteration of each of these estimation algorithms. Both MW and LSMR fail to scale beyond datasets with 10 attributes, as they both require materializing $\boldsymbol{p}$ in vector form, while PrivatePGM easily scales to datasets with 1000 attributes.

While the domain size is the primary factor that determines scalability of the baseline methods MW and LSMR, the scalability of PrivatePGM primarily depends primarily on the measured cliques which determine the structure of the underlying graphical model. In this experiment, the measurements were chosen to highlight a case where the graphical model is "nice" and PrivatePGM scales very well.

Varying structure of cliques In our second scalability experiment, we consider a 100-dimensional dataset with $|\Omega_i| = 10$ for each attribute i. Neither MW or LSMR are capable of running on domains this large, so this experiment is primarily about understanding PrivatePGM. For $k = 1, \dots, 10^3$, we select k size three cliques, and measure the corresponding marginals using the Laplace mechanism. Two procedures are used for selecting the cliques: "random" and "greedy". The random method selects the cliques uniformly at random without replacement, while the greedy method selects the cliques to minimize the size of the resulting junction tree. In Figure 4.2c, we show that PrivatePGM scales much more favorably with greedily selected cliques, which is not surprising because the size of the junction tree determines the complexity of `MARGINAL-ORACLE`. In fact, with greedily selected cliques, PrivatePGM easily scaled to $k = 10^3$, while for randomly selected cliques it only scaled to $k = 51$. This is a huge difference, and thus a crucial property of PrivatePGM that practitioners interested in using it need to be aware of.

4.7 Discussion and limitations

In this chapter, we presented PrivatePGM, a general-purpose technique for estimating a data distribution from differentially private observations. PrivatePGM can be plugged into a number of existing mechanisms for discrete data, immediately improving their utility and/or scalability. PrivatePGM utilizes a compact factored representation of the data vector, which enables it to scale effectively to truly high-dimensional settings. In our experiments, we ran it on synthetic datasets with up to 1000 attributes, and corresponding domain size of $n = 10^{1000}$. There are two main limitations of PrivatePGM, which are explained below.

First, PrivatePGM is not an end-to-end mechanism, but rather a tool for post-processing the output of an existing mechanism. It plugs in naturally to a variety of existing mechanisms, but can not be run "in isolation" without noisy observations as input. This is not a limitation, but an intentional design decision. Since PrivatePGM provides a principled solution to the `reconstruct` subproblem, it allows researchers to focus their energy on the equally important and orthogonal `select` subproblem. Since it was originally published, PrivatePGM has been used in exactly this manner as a core component of three published mechanisms for synthetic data generation: MST [70], PrivMRF [16], and AIM (Chapter 5).

Second, PrivatePGM is capable of scaling to high-dimensional settings only when the measurements "allow it". That is, the scalability of PrivatePGM depends on the cliques that define the graphical model, which is determined by which marginals the mechanism measured. As we saw in Section 4.6, there is a big difference in runtime when these cliques are chosen greedily and when they are chosen randomly. Furthermore, it requires some expertise in graphical models to understand the nuances to the scalability of PrivatePGM. To help non-experts use PrivatePGM effectively, the open source implementation exposes a function JT-SIZE $(r_1, \dots, r_k)$ which consumes a list of cliques and returns the size of the corresponding junction tree. The complexity of PrivatePGM is closely related to JT-SIZE, and if it is sufficiently small, PrivatePGM can be expected to run efficiently. There are two natural ways to overcome this limitation, one is to design approximations that relax the optimization problem underlying PrivatePGM in some way, and the other is to design a mechanism that carefully selects cliques to avoid this worst-case behavior by construction. We developed the former idea in [72], and develop the latter idea in Chapter 5.

CHAPTER 5

AIM: AN ADAPTIVE AND ITERATIVE MECHANISM FOR DIFFERENTIALLY PRIVATE SYNTHETIC DATA

5.1 Motivation

In this chapter, we consider the problem of differentially private synthetic data generation. This problem calls for generating a collection of records matching the input schema, intended to be broadly representative of the source data, in a differentially private manner. Private synthetic data is appealing because it fits any data processing workflow designed for the original data, and, on its face, the user may believe they can perform *any* computation they wish, while still enjoying the benefits of privacy protection. Unfortunately it is well-known that there are limits to the accuracy that can be provided by synthetic data, under differential privacy or any other reasonable notion of privacy [27]. As a consequence, it is important to tailor synthetic data to the intended *workload*.

While the problem of differentially private synthetic data has received considerable research attention [110, 20, 113, 103, 102, 91, 95, 62, 92, 19, 38, 48, 50, 115, 89, 1, 10, 116, 4, 61], only a small subset of the prior work considered the concept of a workload as part of the problem statement [37, 95, 5, 64]. While workload-awareness is a useful distinguishing characteristic of these mechanisms, our experiments will reveal that existing workload-aware mechanisms often fail to outperform workload-agnostic mechanisms, *even when evaluated specifically on their target workloads.* Not only do these algorithms fail to produce accurate synthetic data, they provide no way for end-users to detect the inaccuracy. As a result, in practical terms, differentially private synthetic data generation remains an unsolved problem.

In this work, we advance the state-of-the-art of differentially private synthetic data in two key ways. First, we propose a novel workload-aware mechanism that offers lower error than all competing techniques. Second, we derive analytic expressions to bound the per-query error of the mechanism with high probability.

Our mechanism, AIM, follows the `select-measure-reconstruct` paradigm introduced in Chapter 3. We leverage PrivatePGM [73] for the reconstruct step, as it provides a robust and efficient method for combining the noisy measurements into a single consistent estimate of the data distribution from which synthetic records can be generated.

The low error of AIM is primarily due to innovations in the `select` stage. AIM uses an iterative, greedy selection procedure, inspired by the popular MWEM algorithm for linear query answering. Like MWEM, AIM iteratively selects marginals to measure using a carefully designed quality score function. This quality score takes into account: (i) how well the candidate marginal is already estimated, (ii) the expected improvement measuring it can offer, (iii) the relevance of the marginal to the workload, and (iv) the available privacy budget. This novel quality score is accompanied by a host of other algorithmic techniques including adaptive selection of rounds and budget-per-round, intelligent initialization, and novel set of candidates from which to select.

In conjunction with AIM, we develop new techniques to quantify uncertainty in query answers derived from the generated synthetic data. The problem of error quantification for data independent mechanisms like the Laplace or Gaussian mechanism is trivial, as they provide unbiased answers with known variance to all queries. The problem is considerably more challenging for data-dependent mechanisms like AIM, where complex post-processing is performed and only a subset of workload queries have unbiased answers. Some mechanisms, like MWEM, provide theoretical guarantees on their worst-case error, under suitable assumptions. However, this is an *a priori* bound on error obtained from a theoretical analysis of the mechanism under worst-case datasets. Instead, we develop an *a posteriori* error analysis, derived from the intermediate differentially private measurements used to produce the synthetic data. Our error estimates therefore reflect the actual execution of AIM on the input data, but do not require any additional privacy budget for their calculation. Formally, our guarantees represent one-sided confidence intervals, and we refer to them simply as "confidence bounds". To our knowledge, AIM is the only differentially private synthetic data generation mechanism that provides this kind of error quantification.

Organization In Section 5.2, we define the problem and assumptions of this work. In Section 5.3, we assess the prior work in the field, characterizing different approaches via key distinguishing elements and limitations.In Section 5.4, we present the core ingredients of AIM. In Section 5.5, we derive analytic expressions to bound the per-query error of AIM with high probability, which we use to construct confidence bounds. In Section 5.6, we experimentally evaluate AIM.

5.2 Problem setup

In this chapter, our goal is to design a mechanism $\mathcal{A} : \mathcal{D} \rightarrow \mathcal{D}$ that consumes a discrete dataset D, and returns a synthetic dataset $\hat{D}$ that conforms to the same domain as D. The quality of the synthetic data will be judged based on a workload, which for simplicity, we assume contains a collection of weighted marginal queries.

Definition 26 (Workload Error)**.** *A workload W consists of a list of marginal queries $r_1, \dots, r_k$ where $r_i \subseteq [d]$, together with associated weights $c_i \geq 0$. The error of a synthetic dataset $\hat{D}$ is defined as:*

$$Error(D, \hat{D}) = \frac{1}{k \cdot |D|} \sum_{i=1}^{k} c_i \left\| M_{r_i}(D) - M_{r_i}(\hat{D}) \right\|_1$$

While this workload class is simpler than the ones considered in the previous chapters, it is still fairly expressive. The formal problem statement we consider in this chapter is stated below:

Problem 5 (Workload Error Minimization)**.** *Given a workload W, design an (ϵ, δ)-DP synthetic data mechanism $\mathcal{M} : \mathcal{D} \rightarrow \mathcal{D}$ such that the expected error defined in Definition 26 is minimized.*

5.3 Prior work on synthetic data

In this section we survey the state of the field, describing basic elements of a good synthetic data mechanism, along with novelties of more sophisticated mechanisms. We focus our attention on *marginal-based approaches* to differentially private synthetic data in this section, as these have generally seen the most success in practical applications. These mechanisms include PrivBayes [110], PrivBayes+PGM [73], MWEM+PGM [73], MST [70], PrivSyn [116], RAP [5], GEM [64], and PrivMRF [16].

5.3.1 The `select-measure-reconstruct` paradigm

We begin by providing a broad overview of the basic approach employed by many differentially private mechanisms for synthetic data. These mechanisms all fit naturally into the `select-measure-reconstruct` framework. Recall from Chapter 3 that this framework represents a class of mechanisms which can naturally be broken up into 3 steps: (1) `select` a set of queries, (2) `measure` those queries using a noise-addition mechanism, and (3) `reconstruct` synthetic data that explains the noisy measurements well. We consider iterative mechanisms that alternate between the select and measure step to be in this class as well. Mechanisms within this class differ in their methodology for selecting queries, the noise mechanism used, and the approach to generating synthetic data from the noisy measurements.

Algorithm 5 MWEM+PGM

```
Input: Dataset $D$, workload $W$, privacy parameter $\rho$, rounds $T$
Output: Synthetic Dataset $\hat{D}$
$\epsilon = 2\sqrt{\rho/T}$
$\sigma = \sqrt{T/\rho}$
$\boldsymbol{\theta}_0 = \mathbf{0}$
for $t = 1, \dots, T$ do
    select $r_t \in W$ using the exponential mechanism with $\epsilon$ budget and score function:
        $q_r(D) = \left\| M_r(D) - M_r(\boldsymbol{p}_{\boldsymbol{\theta}_{t-1}}) \right\|_1 - n_r$
    measure marginal on $r_t$ with the Gaussian mechanism with scale $\sigma$:
        $\tilde{\boldsymbol{\mu}}_{r_t} = M_{r_t}(D) + \mathcal{N}(0, \sigma^2)^{n_{r_t}}$
    reconstruct data distribution $\boldsymbol{p}_{\boldsymbol{\theta}_t}$ using PrivatePGM with loss function:
        $L(\boldsymbol{\mu}) = \sum_{j=1}^{t} \left\| \boldsymbol{\mu}_{r_j} - \tilde{\boldsymbol{\mu}}_{r_j} \right\|_2^2$
end for
reconstruct synthetic data $\hat{D}$ from $\boldsymbol{p}_{\boldsymbol{\theta}_T}$ using PrivatePGM
return $\hat{D}$
```

MWEM+PGM, shown in Algorithm 5, is one mechanism from this class that serves as a concrete example as well as the starting point for our improved mechanism, AIM. As the name implies, MWEM+PGM is a scalable instantiation of the well-known MWEM algorithm [42] for linear query answering, where the multiplicative weights (MW) step is replaced by a call to PrivatePGM. It is a greedy, iterative mechanism for workload-aware synthetic data generation, and there are several variants. One variant is shown in Algorithm 5. The mechanism begins by initializing an estimate of the joint distribution to be uniform over the data domain ($\boldsymbol{\theta} = 0$). Then, it runs for T rounds, and in each round it does three things: (1) `select` (via the exponential mechanism) a marginal query that is poorly approximated under the current estimate, (2) `measure` the selected marginal using the Gaussian mechanism, and (3) `reconstruct` a new data distribution (using PrivatePGM) that explains the noisy measurements well. After T rounds, the estimated distribution is used to generate synthetic tabular data. In the subsequent subsections, we will characterize existing mechanisms in terms of how they approach these different aspects of the problem.

5.3.2 Basic elements of a good mechanism

In this section we outline some basic criteria reasonable mechanisms should satisfy to get good performance. These recommendations primarily apply to the `measure` step.

Measure entire marginals Marginals are an appealing statistic to measure because every individual contributes a count of one to exactly one cell of the marginal. As a result, we can measure every cell of $M_r(D)$ at the same privacy cost of measuring a single cell. With a few exceptions [5, 64, 95], existing mechanisms utilize this property of marginals or can be extended to use it. The alternative of measuring a single counting query at a time sacrifices utility unnecessarily.

Use Gaussian noise. Back of the envelope calculations reveal that if the number of measurements is greater than roughly $\log(1/\delta) + \epsilon$, which is often the case, then the standard deviation of the required Gaussian noise is lower than that of the Laplace noise. Many newer mechanisms recognize this and use Gaussian noise, while older mechanisms were developed with Laplace noise, but can easily be adapted to use Gaussian noise instead.

Use unbounded DP For fixed (ϵ, δ), the required noise magnitude is lower by a factor of $\sqrt{2}$ when using unbounded DP (add / remove one record) over bounded DP (modify one record). This is because the L_2 sensitivity of a marginal query M_r is 1 under unbounded DP, and $\sqrt{2}$ under bounded DP. Some mechanisms like MST, PrivSyn, and PrivMRF use unbounded DP, while other mechanisms like RAP, GEM,

Table 5.1: Taxonomy of mechanisms in the `select-measure-reconstruct` paradigm.

Name	Year	Workload Aware	Data Aware	Budget Aware	Efficiency Aware
Independent	-				✓
Gaussian+PGM	-	✓			
PrivBayes [110]	2014		✓	✓	✓
HDMM+PGM [73]	2019	✓			
PrivBayes+PGM [73]	2019		✓	✓	✓
MWEM+PGM [73]	2019	✓	✓		
PrivSyn [116]	2020		✓	✓	✓
MST [70]	2021		✓		✓
RAP [5]	2021	✓	✓		✓
GEM [64]	2021	✓	✓		✓
PrivMRF [16]	2021		✓	✓	✓
AIM [This Work]	2022	✓	✓	✓	✓

and PrivBayes use bounded DP. We remark that these two different definitions of DP are qualitatively different, and because of that, the privacy parameters have different interpretations. The $\sqrt{2}$ difference could be recovered in bounded DP by increasing the privacy budget appropriately.

5.3.3 Distinguishing elements of existing work

Beyond the basics, different mechanisms exhibit different novelties, and understanding the design considerations underlying the existing work can be enlightening. We provide a simple taxonomy of this space in Table 5.1 in terms of four criteria: workload-, data-, budget-, and efficiency-awareness. These characteristics primarily pertain to the `select` step of each mechanism.

Workload-awareness Different mechanisms select from a different set of candidate marginal queries. PrivBayes and PrivMRF, for example, select from a particular subset of k-way marginals, determined from the data. Other mechanisms, like MST and PrivSyn, restrict the set of candidates to 2-way marginal queries. On the other end of the spectrum, the candidates considered by MWEM+PGM, RAP, and GEM, are exactly the marginal queries in the workload. This is appealing, since these mechanisms will not waste the privacy budget to measure marginals that are not relevant to the workload.

Data-awareness Many mechanisms select marginal queries from a set of candidates based on the data, and are thus data-aware. For example, MWEM+PGM selects marginal queries using the exponential mechanism with a quality score function that depends on the data. Independent, Gaussian, and HDMM+PGM are the exceptions, as they always select the same marginal queries no matter what the underlying data distribution is.

Budget-awareness Another aspect of different mechanisms is how well do they adapt to the privacy budget available. Some mechanisms, like PrivBayes, PrivSyn, and PrivMRF recognize that we can afford to measure more (or larger) marginals when the privacy budget is sufficiently large. When the privacy budget is limited, these mechanisms recognize that fewer (and smaller) marginals should be measured instead. In contrast, the number and size of the marginals selected by mechanisms like MST, MWEM+PGM, RAP, and GEM does not depend on the privacy budget available.[1]

Efficiency-awareness Mechanisms that build on top of PrivatePGM must take care when selecting measurements to ensure JT-SIZE remains sufficiently small to ensure computational tractability. Among these, PrivBayes+PGM, MST, and PrivMRF all have built-in heuristics in the selection criteria to ensure the selected marginal queries give rise to a tractable model. Gaussian, HDMM+PGM and MWEM+PGM have no such safeguards, and they can sometimes select marginal queries that lead to intractable models. In the extreme case, when the workload is all 2-way marginals,

[1]The number of rounds to run MWEM+PGM, RAP, and GEM is a hyper-parameter, and the best setting of this hyper-parameter depends on the privacy budget available.

Gaussian selects all 2-way marginals, model required for PrivatePGM explodes to the size of the entire domain, which is often intractable.

Mechanisms that utilize different techniques for post-processing noisy marginals into synthetic data, like PrivSyn, RAP, and GEM, do not have this limitation, and are free to select from a wider collection of marginals. While these methods do not suffer from this particular limitation of PrivatePGM, they have other pros and cons which were surveyed in a recent article [68].

Summary With the exception of our new mechanism AIM, no mechanism listed in Table 5.1 is aware of all four factors we discussed. Mechanisms that do not have four checkmarks in Table 5.1 are not necessarily bad, but there are clear ways in which they can be improved. Conversely, mechanisms that have more checkmarks than other mechanisms are not necessarily better. For example, RAP has 3 checkmarks, but as we show in Section 5.6, it does not consistently beat Independent, which only has 1 checkmark.

5.3.4 Other design considerations

Beyond these four characteristics summarized in the previous section, different methods make different design decisions that are relevant to mechanism performance, but do not correspond to the four criteria discussed in the previous section. In this section, we summarize some of those additional design considerations.

Selection method Some mechanisms select marginals to measure in a *batch*, while other mechanisms select them *iteratively*. Generally speaking, iterative methods like MWEM+PGM, RAP, GEM, and PrivMRF are preferable to batch methods, because the selected marginals will capture important information about the distribution that was not effectively captured by the previously measured marginals. On the other hand, PrivBayes, MST, and PrivSyn select all the marginals before measuring any of them. It is not difficult to construct examples where a batch method like PrivSyn has suboptimal behavior. For example, suppose the data contains three perfectly correlated attributes. We can expect iterative methods to capture the distribution after measuring any two 2-way marginals. On the other hand, a batch method like PrivSyn will determine that all three 2-way marginals need to be measured.

Budget split Every mechanism in this discussion, except for PrivSyn, splits the privacy budget equally among selected marginals. This is a simple and natural thing to do, but it does not account for the fact that larger marginals have smaller counts that are less robust to noise, requiring a larger fraction of the privacy budget to answer accurately. PrivSyn provides a simple formula for dividing privacy budget among marginals of different sizes, but this approach is inherently tied to their batch selection methodology. It is much less clear how to divide the privacy budget within a mechanism that uses an iterative selection procedure.

Hyperparameters All mechanisms have some hyperparameters than can be tuned to affect the behavior of the mechanism. Mechanisms like PrivBayes, MST, PrivSyn, and PrivMRF have reasonable default values for these hyperparameters, and these mechanisms can be expected to work well out of the box. On the other hand, MWEM+PGM, RAP, and GEM have to tune the number of rounds to run, and it is not obvious how to select this a priori. While the open source implementations may include a default value, the experiments conducted in the respective papers did not use these default values, in favor of non-privately optimizing over this hyper-parameter for each dataset and privacy level considered [5, 64].

Algorithm 6 AIM: An Adaptive and Iterative Mechanism

```
1: **Input:** Dataset $D$, workload $W$, privacy parameter $\rho$
2: **Output:** Synthetic Dataset $\hat{D}$
3: **Hyper-Parameters:** MAX-SIZE=80MB, $T = 16d$, $\alpha = 0.9$
4: $\sigma_0 = \sqrt{T/(2\,\alpha\,\rho)}$
5: $\rho_{used} = 0$
6: $t = 0$
7: Initialize $\boldsymbol{p}_{\boldsymbol{\theta}_t}$ using Algorithm 7
8: $w_r = \sum_{s \in W} c_s \mid r \cap s \mid$
9: $\sigma_{t+1} \leftarrow \sigma_0 \quad \epsilon_{t+1} \leftarrow \sqrt{8(1-\alpha)\rho/T}$
10: **while** $\rho_{used} < \rho$ **do**
11:     $t = t + 1$
12:     $\rho_{used} \leftarrow \rho_{used} + \frac{1}{8}\epsilon_t^2 + \frac{1}{2\sigma_t^2}$
13:     $C_t = \{r_t \in W_+ \mid \text{JT-SIZE}(r_1, \dots, r_t)) \leq \frac{\rho_{used}}{\rho} \cdot \text{MAX-SIZE}\}$
14:     select $r_t \in C_t$ using the exponential mechanism $\epsilon$ budget and score function:
            $$q_r(D) = w_r\Big( \left\|M_r(D) - M_r(\boldsymbol{p}_{\boldsymbol{\theta}_{t-1}})\right\|_1 - \sqrt{2/\pi} \cdot \sigma_t \cdot n_r\Big)$$
15:     measure marginal on $r_t$ using the Gaussian mechanism with scale $\sigma$:
            $$\tilde{\boldsymbol{\mu}}_{r_t} = M_{r_t}(D) + \mathcal{N}(0, \sigma_t^2)^{n_{r_t}}$$
16:     reconstruct the data distribution using PrivatePGM with loss function:
            $$L(\boldsymbol{\mu}) = \sum_{j=1}^{t} \frac{1}{\sigma_j} \left\|\boldsymbol{\mu}_{r_j} - \tilde{\boldsymbol{\mu}}_{r_j}\right\|_2^2$$
17:     anneal $\epsilon_{t+1}$ and $\sigma_{t+1}$ using Algorithm 8
18: **end while**
19: reconstruct synthetic data $\hat{D}$ from $\boldsymbol{p}_{\boldsymbol{\theta}_t}$ using PrivatePGM
20: **return** $\hat{D}$
```

Algorithm 7 Initialize $\boldsymbol{p}_{\boldsymbol{\theta}_t}$ (subroutine of Algorithm 6)

```
1: **for** $r \in \{r \in W_+ \mid |r| = 1\}$ **do**
2:     $t = t + 1 \quad \sigma_t \leftarrow \sigma_0 \quad r_t \leftarrow r$
3:     $\tilde{\boldsymbol{\mu}}_{r_t} = M_{r_t}(D) + \mathcal{N}(0, \sigma_t^2)^{r_t}$
4:     $\rho_{used} \leftarrow \rho_{used} + \frac{1}{2\sigma_t^2}$
5: **end for**
6: reconstruct $\boldsymbol{p}_{\boldsymbol{\theta}_t}$ using PrivatePGM with $L(\boldsymbol{\mu}) = \sum_{j=1}^{t} \frac{1}{\sigma_j} \left\|\boldsymbol{\mu}_{r_j} - \tilde{\boldsymbol{\mu}}_{r_j}\right\|_2^2$
```

Algorithm 8 Budget annealing (subroutine of Algorithm 6)

```
1: if $\left\|M_{r_t}(\boldsymbol{p}_{\boldsymbol{\theta}_t}) - M_{r_t}(\boldsymbol{p}_{\boldsymbol{\theta}_{t-1}})\right\|_1 \leq \sqrt{2/\pi} \cdot \sigma_t \cdot n_{r_t}$ then
2:     $\epsilon_{t+1} \leftarrow 2 \cdot \epsilon_t$
3:     $\sigma_{t+1} \leftarrow \sigma_t / 2$
4: else
5:     $\epsilon_{t+1} \leftarrow \epsilon_t$
6:     $\sigma_{t+1} \leftarrow \sigma_t$
7: end if
8: if $(\rho - \rho_{used}) \leq 2\left(\frac{1}{2\sigma^2_{t+1}} + \frac{1}{8}\epsilon^2_{t+1}\right)$ then
9:     $\epsilon_{t+1} = \sqrt{8 \cdot (1-\alpha) \cdot (\rho - \rho_{used})}$
10:    $\sigma_{t+1} = \sqrt{1/(2 \cdot \alpha \cdot (\rho - \rho_{used}))}$
11: end if
```

5.4 Mechanism components

While MWEM+PGM is a simple and intuitive algorithm, it leaves significant room for improvement. Our new mechanism, AIM, is presented in Algorithm 6. In this section, we describe the differences between MWEM+PGM and AIM, the justifications for the relevant design decisions, as well as prove the privacy of AIM.

Intelligent Initialization. In Line 7 of AIM, we spend a small fraction of the privacy budget to measure 1-way marginals in the set of candidates. Estimating $\boldsymbol{p}_{\boldsymbol{\theta}}$ from these noisy marginals gives rise to an *independent* model where all 1-way marginals are preserved well, and higher-order marginals can be estimated under an independence assumption. This provides a far better initialization than the default uniform distribution while requiring only a small fraction of the privacy budget.

New Candidates. In Line 13 of AIM, we make two notable modifications to the candidate set that serve different purposes. Specifically, the set of candidates is a carefully chosen subset of the marginal queries in the *downward closure* of the workload. The downward closure of the workload is the set of cliques that are subsets of some clique in the workload, i.e., $W_+ = \{r \mid r \subseteq s, s \in W\}$.

Using the downward closure is based on the observation that marginals with many attributes have low counts, and answering them directly with a noise addition mechanism may not provide an acceptable signal to noise ratio. In these situations, it may be better to answer lower-dimensional marginals, as these tend to exhibit a better signal to noise ratio, while still being useful to estimate the higher-dimensional marginals in the workload.

We filter candidates from this set that do not meet a specific model capacity requirement. Specifically, the set will only consist of candidates that, if selected, will lead to a JT-SIZE below a prespecified limit (the default is 80 MB). This ensures that AIM will never select candidates that lead to an intractable model for PrivatePGM, and hence allows the mechanism to execute consistently with a predictable memory footprint and runtime.

Better Selection Criteria. In Line 14 of AIM, we make two modifications to the quality score function for marginal query selection to better reflect the utility we expect from measuring the selected marginal. In particular, our new quality score function is

$$q_r(D) = w_r\big(\big\| M_r(D) - M_r(\boldsymbol{p}_{\boldsymbol{\theta}_{t-1}}) \big\|_1 - \sqrt{2/\pi} \cdot \sigma_t \cdot n_r \big), \tag{5.1}$$

which differs from MWEM+PGM's quality score function $q_r(D) = \big\| M_r(D) - M_r(\boldsymbol{p}_{\boldsymbol{\theta}_{t-1}}) \big\| - n_r$ in two ways.

First, the expression inside parentheses can be interpreted as the *expected improvement* in L_1 error we can expect by measuring that marginal. It consists of two terms: the L_1 error under the current model minus the expected L_1 error if it is measured at the current noise level. Compared to the quality score function in MWEM+PGM, this quality score function penalizes larger marginals to a much more significant degree, since $\sigma_t \gg 1$ in most cases. Moreover, this modification makes the selection criteria "budget-adaptive", since it recognizes that we can afford to measure larger marginals when σ_t is smaller, and we should prefer smaller marginals when σ_t is larger.

Second, we give different marginal queries different weights to capture how relevant they are to the workload. In particular, we weight the quality score function for a marginal query r using the formula $w_r = \sum_{s \in W} c_s \mid r \cap s \mid$, as this captures the degree to which the marginal queries in the workload overlap with r. In general, this weighting scheme places more weight on marginals involving more attributes. Note that now the sensitivity of q_r is w_r rather than 1. When applying the exponential mechanism to select a candidate, we must either use $\Delta_t = \max_{r \in C_t} w_r$, or invoke the generalized exponential mechanism instead, as it can handle quality score functions with varying sensitivity [85].

This quality score function exhibits an interesting trade-off: the penalty term $\sqrt{2/\pi}\sigma_t n_r$ discourages marginals with more cells, while the weight w_r favors marginals with more attributes. However, if the inner expression is negative, then the larger weight will make it more negative, and much less likely to be selected.

Adaptive Rounds and Budget Split. In Lines 12 and 17 of AIM, we introduce logic to modify the per-round privacy budget as execution progresses, and as a result, eliminate the need to provide the number of rounds up front. This makes AIM hyper-parameter free, relieving practitioners from that often overlooked burden.

Specifically, we use a simple annealing procedure (Algorithm 8) that gradually increases the budget per round when an insufficient amount of information is learned at the current per-round budget. The annealing condition is activated if the difference between $M_{r_t}(\boldsymbol{p}_{\boldsymbol{\theta}_t})$ and $M_{r_t}(\boldsymbol{p}_{\boldsymbol{\theta}_{t-1}})$ is small, which indicates that not much information was learned in the previous round. If it is satisfied, then ϵ_t for the `select` step is doubled, while σ_t for the `measure` step is cut in half.

This check can pass for two reasons: (1) there were no good candidates (all scores are low in Equation (5.1)) in which case increasing σ_t will make more candidates good, and (2) there were good candidates, but they were not selected because there was too much noise in the select step, which can be remedied by increasing ϵ_t. The precise annealing threshold used is $\sqrt{2/\pi} \cdot \sigma_t \cdot n_{r_t}$, which is the expected error of the noisy marginal, and an approximation for the expected error of $\boldsymbol{p}_{\boldsymbol{\theta}_t}$ on marginal r. When the available privacy budget is small, this condition will be activated more frequently, and as a result, AIM will run for fewer rounds. Conversely, when the available privacy budget is large, AIM will run for many rounds before this condition activates.

As σ_t decreases throughout execution, quality scores generally increase, and it has the effect of "unlocking" new candidates that previously had negative quality scores. We initialize σ_t and ϵ_t conservatively, assuming the mechanism will be run for $T = 16d$ rounds. This is an upper bound on the number of rounds that AIM will run, but in practice the number of rounds will be much less.

As in prior work [116, 16], we do not split the budget equally for the select and measure step, but rather allocate 10% of the budget for the select steps, and 90% of the budget for the measure steps. This is justified by the fact that the quality function for selection is a coarser-grained aggregation than a marginal, and as a result can tolerate a larger degree of noise.

Privacy Analysis. The privacy analysis of AIM utilizes the notion of a *privacy filter* [88], and the algorithm runs until the realized privacy budget spent matches the total privacy budget available, ρ. To ensure that the budget is not over-spent, there is a special condition (Line 8 in Algorithm 8) that checks if the remaining budget is insufficient for two rounds at the current ϵ_t and σ_t parameters. If this condition is satisfied, ϵ_t and σ_t are set to use up all of the remaining budget in one final round of execution.

Theorem 13. *For any $T \geq d$, $0 < \alpha < 1$, and $\rho \geq 0$, AIM satisfies ρ-zCDP.*

Proof. There are three steps in AIM that depend on the sensitive data: initialization, selection, and measurement. The initialization step satisfies ρ_0-zCDP for $\rho_0 = |\{r \in W_+ \mid |r| = 1\}|/2\sigma_0^2 \leq d/2\sigma_0^2 = 2\alpha d\rho/2T \leq \rho$. For this step, all we need is that the privacy budget is not over-spent. The remainder of AIM runs until the budget is consumed. Each step of AIM involves one invocation of the exponential mechanism, and one invocation of the Gaussian mechanism. By Propositions 4 to 6, round t of AIM is ρ_t-zCDP for $\rho_t = \frac{1}{8}\epsilon_t^2/8 + 1/2\sigma_t^2$. Note that at round t, $\rho_{used} = \sum_{i=0}^{t} \rho_i$, and we need to show that ρ_{used} never exceeds ρ [88]. There are two cases to consider: the condition in Line 8 of Algorithm 8 is either true or false. If it is true, then we know after round t that $\rho - \rho_{used} \geq 2\rho_{t+1}$, i.e., the remaining budget is enough to run round $t+1$ without over-spending the budget. If it is false, then we modify ϵ_{t+1} and ρ_{t+1} to exactly use up the remaining budget. Specifically, $\rho_{t+1} = 8(1-\alpha)(\rho - \rho_{used})/8 + 2\alpha(\rho - \rho_{used})/2 = \rho - \rho_{used}$. As a result, when the condition is true, ρ_{used} at time $t+1$ is exactly ρ, and after that iteration, the main loop of AIM terminates. The remainder of the mechanism does not access the data. □

5.5 Uncertainty quantification

In this section, we propose a solution to the uncertainty quantification problem for AIM. Our method uses information from *both* the noisy marginals, measured with Gaussian noise, and the marginal queries selected by the exponential mechanism. Importantly, the method does not require additional privacy budget, as it quantifies uncertainty only by analyzing the private outputs of AIM. We give guarantees for marginals in the (downward closure of the) workload, which is exactly the set of marginals the analyst cares about. We provide no guarantees for marginals outside this set, which is an area for future work.

We break our analysis up into two cases: the "easy" case, where we have access to unbiased answers for a particular marginal, and the "hard" case, where we do not.

In both cases, we identify an *estimator* for a marginal whose error we can bound with high probability. Then, we connect the error of this estimator to the error of the synthetic data by invoking the triangle inequality. The subsequent paragraphs provide more details on this approach. Proofs of all statements in this section appear in the full paper [71].

The easy case: supported marginal queries A marginal query r is "supported" whenever $r \subseteq r_t$ for some t. In this case, we can readily obtain an unbiased estimate of $M_r(D)$ from $\tilde{\boldsymbol{\mu}}_{r_t}$, and analytically derive the variance of that estimate. If there are multiple t satisfying the condition above, we have multiple estimates we can use to reduce the variance. We can combine these independent estimates to obtain a *weighted average estimator*:

Theorem 14 (Weighted Average Estimator)**.** *Let* $r_1, \dots, r_t$ *and* $\tilde{\boldsymbol{\mu}}_{r_1}, \dots, \tilde{\boldsymbol{\mu}}_{r_t}$ *be as defined in Algorithm 6, and let* $R = \{r_1, \dots, r_t\}$. *For any* $r \in R_+$, *there is an (unbiased) estimator* $\bar{\boldsymbol{\mu}}_r = f_r(\tilde{\boldsymbol{\mu}}_{r_1}, \dots, \tilde{\boldsymbol{\mu}}_{r_t})$ *such that:*

$$\bar{\boldsymbol{\mu}}_r \sim \mathcal{N}(M_r(D), \bar{\sigma}_r^2)^{n_r} \quad \textit{where} \quad \bar{\sigma}_r^2 = \Big[\sum_{\substack{j=1 \\ r \subseteq r_j}}^{t} \frac{n_r}{n_{r_j} \sigma_j^2} \Big]^{-1},$$

Proof. For each $r_j \supseteq r$, we observe $\tilde{\boldsymbol{\mu}}_{r_j} \sim M_{r_j}(D) + \mathcal{N}(0, \sigma_i^2)^{n_{r_j}}$. We can use this noisy marginal to obtain an unbiased estimate $M_r(D)$ by marginalizing out attributes in the set $r_j \setminus r$. This requires summing up n_{r_j}/n_r cells, so the variance in each cell becomes $n_{r_j}\sigma_i^2/n_r$. Moreover, the noise is still normally distributed, since the sum of independent normal random variables is normal. We thus have such an estimate for each i satisfying $r_j \supseteq r$, and we can combine these independent estimates using *inverse variance weighting* [45], resulting in an unbiased estimator with the stated variance. For the same reason as before, the noise is still normally distributed. □

While this is not the only (or best) estimator to use,[2] the simplicity allows us to easily bound its error, as we show in Theorem 15.

Theorem 15 (Confidence Bound)**.** *Let $\bar{\boldsymbol{\mu}}_r$ be the estimator from Theorem 14. Then, for any $\lambda \geq 0$, with probability at least $1 - \exp(-\lambda^2)$:*

$$\|M_r(D) - \bar{\boldsymbol{\mu}}_r\|_1 \leq \sqrt{2\log 2}\bar{\sigma}_r n_r + \lambda\bar{\sigma}_r\sqrt{2n_r}$$

Proof. Noting that $M_r(D) - \bar{\boldsymbol{\mu}}_r \sim \mathcal{N}(0, \sigma^2)^{n_r}$, the statement is a direct consequence of Lemma 1, below. $\square$

Lemma 1. *Let $\boldsymbol{z} \sim N(0, \sigma^2)^n$, then:*

$$\mathbb{E}[\|\boldsymbol{z}\|_1] = \sqrt{2/\pi}n\sigma$$

and

$$\Pr[\|\boldsymbol{z}\|_1 \geq \sqrt{2\log 2}\sigma n + c\sigma\sqrt{2n}] \leq \exp(-c^2)$$

Proof. First observe that $|\boldsymbol{z}(t)|$ is a sample from a *half-normal* distribution. Thus, $\mathbb{E}[\boldsymbol{z}(t)] = \sqrt{2/\pi}\sigma$. From the linearity of expectation, we obtain $\mathbb{E}[\|\boldsymbol{z}\|_1] = \sqrt{2/\pi}\sigma n$, as desired. For the second statement, we begin by deriving the moment generating function of the random variable $|x_i|$. By definition, we have:

[2] A better estimator would be the *minimum variance linear unbiased estimator*. Ding et al. [26] derive an efficient algorithm for computing this from noisy marginals.

$$
\begin{aligned}
\mathbb{E}[\exp{(t \cdot |\boldsymbol{z}(t)|)}] &= \int_{-\infty}^{\infty} \phi(z) \exp{(t \cdot |z|)} dz \\
&= 2 \int_{0}^{\infty} \phi(z) \exp{(t \cdot z)} dz \\
&= 2 \int_{0}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} \exp\Big(- \frac{z^2}{2\sigma^2}\Big) \exp{(t \cdot z)} dz \\
&= \frac{1}{\sigma}\sqrt{\frac{2}{\pi}} \int_{0}^{\infty} \exp\Big(- \frac{z^2}{2\sigma^2} + t \cdot z\Big) dz \\
&= \exp\Big(\frac{\sigma^2 t^2}{2}\Big)\Big(\Phi\Big(\frac{t\sigma}{\sqrt{2}}\Big) + 1\Big)
\end{aligned}
$$

Moreover, since $\|\boldsymbol{z}\|_1 = \sum_{t=1}^{n} |\boldsymbol{z}(t)|$ is a sum of i.i.d random variables, the moment generating function of $\|\boldsymbol{z}\|_1$ is:

$$
\mathbb{E}[\exp{(t \cdot \|\boldsymbol{z}\|_1)}] = \exp\Big(\frac{\sigma^2 t^2}{2}\Big)^n \Big(\Phi\Big(\frac{t\sigma}{\sqrt{2}}\Big) + 1\Big)^n
$$

From the Chernoff bound, we have

$$
\begin{aligned}
\Pr[\|\boldsymbol{z}\|_1 \geq a] &\leq \min_{t \geq 0} \frac{\mathbb{E}[\exp{(t \cdot \|\boldsymbol{z}\|_1)}]}{\exp{(ta)}} \\
&= \min_{t \geq 0} \exp\Big(\frac{n\sigma^2 t^2}{2} - ta\Big)\Big(\Phi\Big(\frac{t\sigma}{\sqrt{2}}\Big) + 1\Big)^n \\
&\leq \min_{t \geq 0} 2^n \exp\Big(\frac{n\sigma^2 t^2}{2} - ta\Big) \\
&\leq 2^n \exp\Big(\frac{n\sigma^2 (a/n\sigma^2)^2}{2} - (a/n\sigma^2)a\Big) \\
&= 2^n \exp\Big(\frac{a^2}{2n\sigma^2} - \frac{a^2}{n\sigma^2}\Big) \\
&= 2^n \exp\Big(- \frac{a^2}{2n\sigma^2}\Big) \\
&= \exp\Big(- \frac{a^2}{2n\sigma^2} + n\log 2\Big)
\end{aligned}
$$

With some further manipulation of the bound, we obtain:

$$\Pr[\|\boldsymbol{z}\|_1 \geq d\sigma\sqrt{2n}] \leq \exp\Big(-d^2 + n\log 2\Big) \qquad (a = d\sigma\sqrt{2n})$$
$$\Pr[\|\boldsymbol{z}\|_1 \geq (c + \sqrt{n\log 2})\sigma\sqrt{2n}] \leq \exp(-c^2) \qquad (d = c + \sqrt{n\log 2})$$
$$\Pr[\|\boldsymbol{z}\|_1 \geq \sqrt{2\log 2}\sigma n + c\sigma\sqrt{2n}] \leq \exp(-c^2)$$

□

Note that Theorem 15 gives a guarantee on the error of $\bar{\boldsymbol{\mu}}_r$, but we are ultimately interested in the error of $\hat{D}$. Fortunately, it easy easy to relate the two by using the triangle inequality, as shown below:

Corollary 1. *Let $\hat{D}$ be any synthetic dataset, and let $\bar{\boldsymbol{\mu}}_r$ be the estimator from Theorem 14. Then with probability at least $1 - \exp(-\lambda^2)$:*

$$\left\|M_r(D) - M_r(\hat{D})\right\|_1 \leq \left\|M_r(\hat{D}) - \bar{\boldsymbol{\mu}}_r\right\|_1 + \sqrt{2\log 2}\bar{\sigma}_r n_r + \lambda\bar{\sigma}_r\sqrt{2n_r}$$

The LHS is what we are interested in bounding, and we can readily compute the RHS from the output of AIM. The RHS is a random quantity that, with the stated probability, upper bounds the error. When we plug in the realized values we get a concrete numerical bound that can be interpreted as a (one-sided) confidence interval. In general, we expect $M_r(\hat{D})$ to be close to $\bar{\boldsymbol{\mu}}_r$, so the error bound for $\hat{D}$ will not be that much larger than that of $\bar{\boldsymbol{\mu}}_r$.[3]

The hard case: unsupported marginal queries We now shift our attention to the hard case, providing guarantees about the error of different marginals even for unsupported marginal queries (those not selected during execution of AIM). This

[3]From prior experience, we might expect the error of $\hat{D}$ to be *lower* than the error of $\bar{\boldsymbol{\mu}}_r$ [80, 73], so we are paying for this difference by increasing the error bound when we might hope to save instead. Unfortunately, this intuition does not lend itself to a clear analysis that provides better guarantees.

problem is significantly more challenging. Our key insight is that marginal queries *not selected* have relatively low error compared to the marginal queries that were selected by virtue of the exponential mechanism and the quality score function we use. We can easily bound the error of selected queries and relate that to non-selected queries by utilizing the guarantees of the exponential mechanism. In Theorem 16 below, we provide expressions that capture the uncertainty of these marginals with respect to $\boldsymbol{p}_{\boldsymbol{\theta}_{t-1}}$, the iterates of AIM.

Theorem 16 (Confidence Bound). *Let $\sigma_t, \epsilon_t, r_t, C_t, \tilde{\boldsymbol{\mu}}_{r_t}, \boldsymbol{p}_{\boldsymbol{\theta}_t}$ be as defined in Algorithm 6, and let $\Delta_t = \max_{r \in C_t} w_r$. For all $r \in C_t$, with probability at least $1 - e^{-\lambda_1^2/2} - e^{-\lambda_2}$:*

$$\left\| M_r(D) - M_r(\boldsymbol{p}_{\boldsymbol{\theta}_{t-1}}) \right\|_1 \leq w_r^{-1} \big(B_r + \lambda_1 \sigma_t \sqrt{n_{r_t}} + \lambda_2 \frac{2\Delta_t}{\epsilon_t} \big)$$

where

$$B_r = w_{r_t} \underbrace{\left\| M_{r_t}(\boldsymbol{p}_{\boldsymbol{\theta}_{t-1}}) - \boldsymbol{\mu}_{r_t} \right\|_1}_{\text{estimated error on } r_t} + \underbrace{\sqrt{2/\pi}\sigma_t \big(w_r n_r - w_{r_t} n_{r_t} \big)}_{\substack{\text{relationship to} \\ \text{non-selected candidates}}} + \underbrace{\frac{2\Delta_t}{\epsilon_t} \log\left(|C_t|\right)}_{\substack{\text{uncertainty from} \\ \text{exponential mech.}}}$$

Proof. By the guarantees of the exponential mechanism, we know that, with probability at most $e^{-\lambda_2}$, for all $r \in C_t$ we have:

$$q_{r_t} \leq q_r - \frac{2\Delta_t}{\epsilon_t} (\log\left(|C_t|\right) + \lambda_2)$$

Now define $E_r = \left\| M_r(D) - M_r(\boldsymbol{p}_{\boldsymbol{\theta}_{t-1}}) \right\|_1$. Plugging in $q_r = w_r(E_r - \sqrt{2/\pi}\sigma_t n_r)$ and rearranging gives:

$$E_r \geq \frac{w_{r_t}(E_{r_t} - \sqrt{2/\pi}\sigma_t n_{r_t}) + \frac{2\Delta_t}{\epsilon_t}(\log(|C_t|) + \lambda_2)}{w_r} + \sqrt{2/\pi}\sigma_t n_r$$

From Lemma 2, with probability at most $e^{-\lambda_1^2/2}$, we have:

$$\left\| M_{r_t}(\boldsymbol{p}_{\boldsymbol{\theta}_{t-1}}) - \tilde{\boldsymbol{\mu}}_{r_t} \right\|_1 + \lambda_1 \sigma_t \sqrt{n_{r_t}} \leq E_{r_t}$$

Combining these two facts via the union bound, along with some algebraic manipulation, yields the stated result. □

Lemma 2. *Let* $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^k$ *and let* $\boldsymbol{c} = \boldsymbol{b} + \boldsymbol{z}$ *where* $\boldsymbol{z} \sim \mathcal{N}(0, \sigma^2)^n$.

$$\Pr[\|\boldsymbol{a} - \boldsymbol{c}\|_1 \leq \|\boldsymbol{a} - \boldsymbol{b}\|_1 - \lambda\sigma\sqrt{n}] \leq \exp\Big(-\frac{1}{2}\lambda^2\Big)$$

Proof. First note that $|\boldsymbol{a}(j) - \boldsymbol{c}(j)| = |\boldsymbol{a}(j) - \boldsymbol{b}(j) - \boldsymbol{z}(j)|$, which is distributed according to a folded normal distribution with mean $|\boldsymbol{a}(j) - \boldsymbol{b}(j)|$. It is well known [93] that the moment generating function for this random variable is $M_i(t)$, where:

$$\begin{aligned} M_i(t) = \exp\Big(\frac{1}{2}\sigma^2 t^2 + |\boldsymbol{a}(j) - \boldsymbol{b}(j)|t\Big)\Phi(|\boldsymbol{a}(j) - \boldsymbol{b}(j)|/\sigma + \sigma t) \\ + \exp\Big(\frac{1}{2}\sigma^2 t^2 - |\boldsymbol{a}(j) - \boldsymbol{b}(j)|t\Big)\Phi(-|\boldsymbol{a}(j) - \boldsymbol{b}(j)|/\sigma + \sigma t). \end{aligned}$$

Moreover, the moment generating function of $\|\boldsymbol{a} - \boldsymbol{c}\|_1$ is $M(t) = \prod_i M_i(t)$. We will begin by focusing our attention on bounding $M_i(-t)$. For simplicity, let $\mu = |\boldsymbol{a}(j) - \boldsymbol{b}(j)|$. We have:

$$
\begin{aligned}
M_i(-t) = & \exp\Big(\frac{\sigma^2 t^2}{2} - \mu t\Big)\Phi(\mu/\sigma - \sigma t) \\
& + \exp\Big(\frac{\sigma^2 t^2}{2} + \mu t\Big)\Phi(-\mu/\sigma - \sigma t) \\
= & \exp\Big(\frac{\sigma^2 t^2}{2} - \mu t\Big)(1 - \Phi(-\mu/\sigma + \sigma t)) \\
& + \exp\Big(\frac{\sigma^2 t^2}{2} + \mu t\Big)\Phi(-\mu/\sigma - \sigma t) \\
= & \exp\Big(\frac{\sigma^2 t^2}{2} - \mu t\Big) \\
& - \exp\Big(\frac{\sigma^2 t^2}{2} - \mu t\Big)\Phi(-\mu/\sigma + \sigma t) \\
& + \exp\Big(\frac{\sigma^2 t^2}{2} + \mu t\Big)\Phi(-\mu/\sigma - \sigma t) \\
\leq & \exp\Big(\frac{\sigma^2 t^2}{2} - \mu t\Big) \qquad \text{(Lemma 3 below; } a = \sigma t, b = \mu/\sigma\text{)}
\end{aligned}
$$

We are now ready to plug this result into the Chernoff bound, which states:

$$
\begin{aligned}
\Pr[\|\boldsymbol{a} - \boldsymbol{c}\|_1 \leq r] & \leq \min_{t \geq 0} \exp(t \cdot r) M(-t) \\
& \leq \min_{t \geq 0} \exp(t \cdot r) \prod_i \exp\Big(\frac{\sigma^2 t^2}{2} - |\boldsymbol{a}(j) - \boldsymbol{b}(j)| t\Big) \\
& = \min_{t \geq 0} \exp\Big(t \cdot r + \frac{n\sigma^2 t^2}{2} - \|\boldsymbol{a} - \boldsymbol{b}\|_1 t\Big)
\end{aligned}
$$

Setting $r = \|\boldsymbol{a} - \boldsymbol{b}\|_1 - \lambda\sigma\sqrt{n}$ gives the desired result

$$
\begin{aligned}
& \Pr[\|\boldsymbol{a} - \boldsymbol{c}\|_1 \leq \|\boldsymbol{a} - \boldsymbol{b}\|_1 - \lambda\sigma\sqrt{n}] \\
& \quad \leq \min_{t \geq 0} \exp\Big(t \cdot (\|\boldsymbol{a} - \boldsymbol{b}\|_1 - \lambda\sigma\sqrt{n}) + \frac{n\sigma^2 t^2}{2} - \|\boldsymbol{a} - \boldsymbol{b}\|_1 t\Big) \\
& \quad = \min_{t \geq 0} \exp\Big(- t\lambda\sigma\sqrt{n} + \frac{n\sigma^2 t^2}{2}\Big) \\
& \quad \leq \exp(-\lambda^2/2) \qquad \text{(set } t = \lambda/\sigma\sqrt{n}\text{)}
\end{aligned}
$$

□

Lemma 3. *Let $a, b \geq 0$, and let Φ denote the CDF of the standard normal distribution. Then,*

$$\exp\Big(\frac{1}{2}a^2 + ab\Big)\Phi(-a-b) \leq \exp\Big(\frac{1}{2}a^2 - ab\Big)\Phi(a-b)$$

Proof. First observe that:

$$\exp\Big(\frac{1}{2}a^2 + ab\Big)\Phi(-a-b) = \exp\Big(-\frac{1}{2}b^2\Big)\frac{\Phi(-a-b)}{\phi(-a-b)}$$
$$\exp\Big(\frac{1}{2}a^2 - ab\Big)\Phi(a-b) = \exp\Big(-\frac{1}{2}b^2\Big)\frac{\Phi(a-b)}{\phi(a-b)}$$

Since $a, b \geq 0$, we know that $-a-b \leq a-b$. We will now argue that the function $\frac{\Phi(\alpha)}{\phi(\alpha)}$ is monotonically increasing in α, which suffices to prove the desired claim. To prove this, we will observe that this is this quantity is known as the *Mills ratio* [39] for the normal distribution. We know that the Mills ratio is connected to a particular expectation; specifically, if $X \sim \mathcal{N}(0,1)$, then

$$\mathbb{E}[X \mid X < \alpha] = -\frac{\phi(\alpha)}{\Phi(\alpha)}$$

Using this interpretation, it is clear that the LHS (and hence the RHS) is monotonically increasing in α. Since $-\frac{\phi(\alpha)}{\Phi(\alpha)}$ is monotonically increasing, so is $\frac{\Phi(\alpha)}{\phi(\alpha)}$. □

We can readily compute B_r from the output of AIM, and use it to provide a bound on error in the form of a one-sided confidence interval that captures the true error with high probability. While these error bounds are expressed with respect to $\boldsymbol{p}_{\boldsymbol{\theta}_{t-1}}$, they can readily be extended to give a guarantee with respect to $\hat{D}$.

Corollary 2. *Let $\hat{D}$ be any synthetic dataset, and let B_r be as defined in Theorem 16. Then with probability at least $1 - e^{-\lambda_1^2/2} - e^{-\lambda_2}$:*

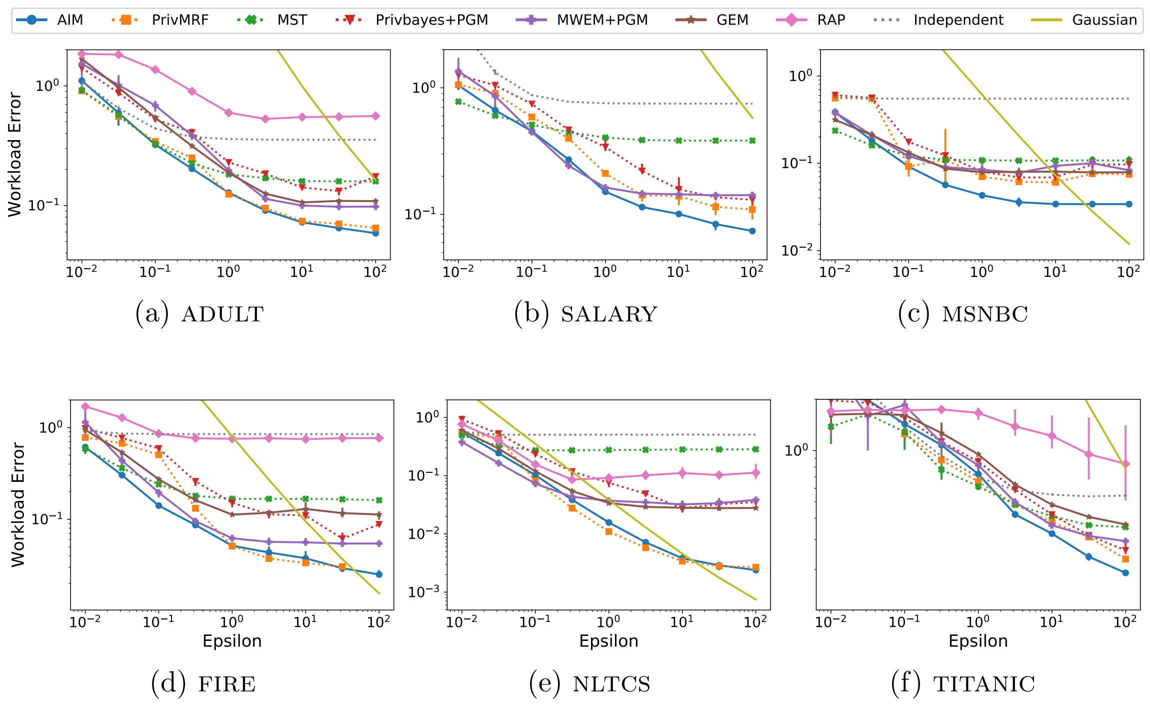
$$\left\| M_r(D) - M_r(\hat{D}) \right\|_1 \leq \left\| M_r(\hat{D}) - M_r(\boldsymbol{p}_{\boldsymbol{\theta}_{t-1}}) \right\|_1 + w_r^{-1} \big(B_r + \lambda_1 \sigma_t \sqrt{n_{r_t}} + \lambda_2 \frac{2\Delta_t}{\epsilon_t} \big)$$

Again, the LHS is what we are interested in bounding, and we can compute the RHS from the output of AIM. We expect $\boldsymbol{p}_{\boldsymbol{\theta}_{t-1}}$ to be reasonably close to $\hat{D}$, especially when t is larger, so this bound will often be comparable to the original bound on $\hat{p}_{\boldsymbol{\theta}_{t-1}}$.

Putting it Together We've provided guarantees for both supported and unsupported marginals. The guarantees for unsupported marginals also apply for supported marginals, although we generally expect them to be looser. In addition, there is one guarantee *for each round of AIM*. It is tempting to use the bound that provides the smallest estimate, although unfortunately doing this invalidates the bound. To ensure a valid bound, we must pick only one round, and that cannot be decided based on the value of the bound. A natural choice is to use only the last round, for three reasons: (1) σ_t is smallest and ϵ_t is largest in that round, (2) the error of $\boldsymbol{p}_{\boldsymbol{\theta}_t}$ generally goes down with t, and (3) the distance between $\boldsymbol{p}_{\boldsymbol{\theta}_t}$ and $\hat{D}$ should be the smallest in the last round. However, there may be some marginal queries which were not in the candidate set for that round. To bound the error on these marginals, we use the last round where that marginal query was in the candidate set.

5.6 Experimental evaluation

In this section we empirically evaluate AIM, comparing it to a collection of state-of-the-art mechanisms and baseline mechanisms for a variety of workloads, datasets, and privacy levels.



(a) ADULT (b) SALARY (c) MSNBC

(d) FIRE (e) NLTCS (f) TITANIC

Figure 5.1: Workload error of competing mechanisms on the ALL-3WAY workload for $\epsilon = 0.01, \dots, 100$.

5.6.1 Experimental setup

Datasets Our evaluation includes datasets with varying size and dimensionality, summarized in the table below.

Table 5.2: Summary of datasets used in the experiments.

Dataset	Records	Dimensions	Min/Max Domains	Total Domain Size
ADULT [52]	48842	15	2–42	4×10^{16}
SALARY [46]	135727	9	3–501	1×10^{13}
MSNBC [15]	989818	16	18	1×10^{20}
FIRE [86]	305119	15	2–46	4×10^{15}
NLTCS [66]	21574	16	2	7×10^{4}
TITANIC [35]	1304	9	2–91	9×10^{7}

Workloads We consider 3 workloads for each dataset, ALL-3WAY, TARGET, and SKEWED. Each workload contains a collection of 3-way marginal queries. The ALL-

3WAY workload contains queries for *all* 3-way marginals. The TARGET workload contains queries for all 3-way marginals involving some specified *target* attribute. For the ADULT and TITANIC datasets, these are the INCOME>50K attribute and the SURVIVED attribute, as those correspond to the attributes we are trying to predict for those datasets. For the other datasets, the target attribute is chosen uniformly at random. The SKEWED workload contains a collection of 3-way marginal queries *biased* towards certain attributes and attribute combinations. In particular, each attribute is assigned a weight sampled from a squared exponential distribution. 256 triples of attributes are sampled with probability proportional to the product of their weights. This results in workloads where certain attributes appear far more frequently than others, and is intended to capture the situation where analysts focus on a small number of interesting attributes. All randomness in the construction of the workload was done with a fixed random seed, to ensure that the workloads remain the same across executions of different mechanisms and parameter settings.

Mechanisms We compare against both workload-agnostic and workload-aware mechanisms in this section. The workload-agnostic mechanisms we consider are PrivBayes+PGM, MST, PrivMRF. The workload-aware mechanisms we consider are MWEM+PGM, RAP, GEM, and AIM. We set the hyper-parameters of every mechanism to default values available in their open source implementations. We also consider baseline mechanisms: Independent and Gaussian. The former measures all 1-way marginals using the Gaussian mechanism, and generates synthetic data using an independence assumption. The latter answers all queries in the workload using the Gaussian mechanism (using the optimal privacy budget allocation described in [116]). Note that this mechanism *does not* generate synthetic data, only query answers.

Privacy Budgets We consider a wide range of privacy parameters, varying $\epsilon \in [0.01, 100.0]$ and setting $\delta = 10^{-9}$. The most practical regime is $\epsilon \in [0.1, 10.0]$, but mechanism behavior at the extremes can be enlightening so we include them as well.

Evaluation. For each dataset, workload, and ϵ, we run each mechanism for 5 trials, and measure the workload error from Definition 26. We report the average workload error across the five trials, along with error bars corresponding to the minimum and maximum workload error observed across the five trials.

Runtime Environment. We ran most experiments on a single core of a compute cluster with a 4 GB memory limit and a 24 hour time limit.[4] These resources were not sufficient to run PrivMRF or RAP, so we utilized different machines to run those mechanisms. PrivMRF requires a GPU to run, so we used one node a different compute cluster, which has a Nvidia GeForce RTX 2080 Ti GPU. RAP required significant memory resources, so we ran those experiments on a machine with 16 cores and 64 GB of RAM.

5.6.2 ALL-3WAY workload

Results on the ALL-3WAY workload are shown in Figure 5.1. Workload-aware mechanisms are shown by solid lines, while workload-agnostic mechanisms are shown with dotted lines. From these plots, we make the following observations:

1. AIM consistently achieves competitive workload error, across all datasets and privacy regimes considered. On average, across all six datasets and nine privacy parameters, AIM improved over PrivMRF by a factor of 1.3×, MST by a factor of 8.4×, MWEM+PGM by a factor 2.1×, PrivBayes+PGM by a factor 2.6×, RAP by a factor 9.5×, and GEM by a factor 2.3×. In the most extreme cases, AIM

[4]These experiments usually completed in well under the time limit.

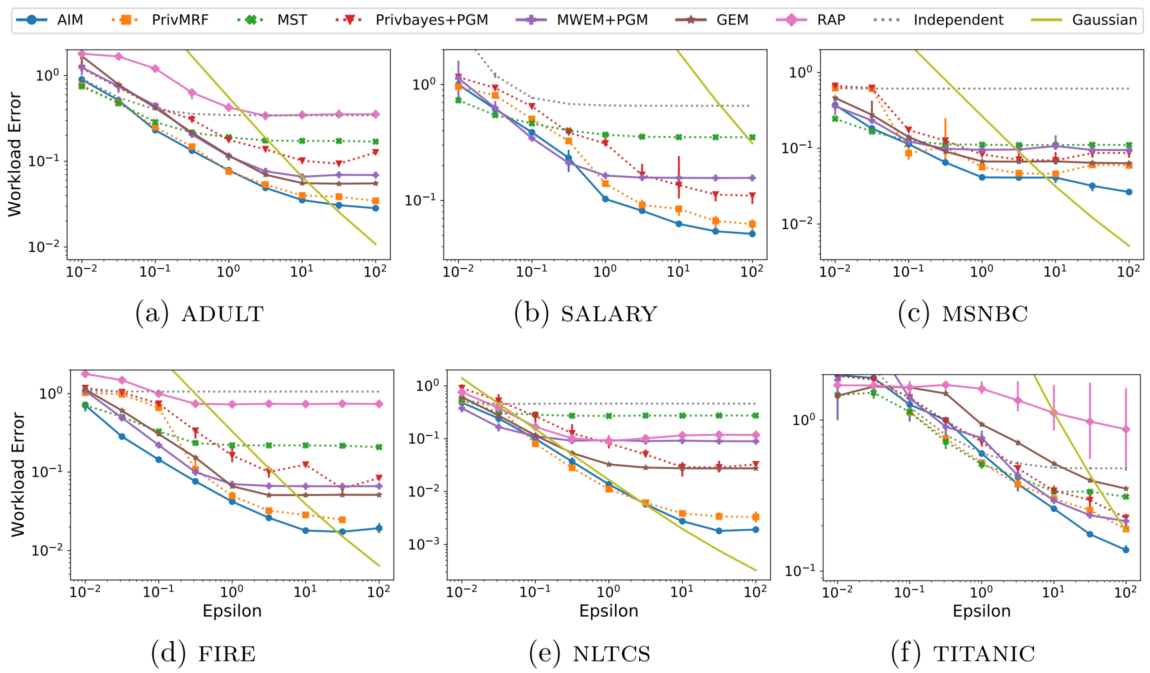


Figure 5.2: Workload error of competing mechanisms on the TARGET workload for $\epsilon = 0.01, \ldots, 100$.

improved over PrivMRF by a factor 3.6×, MST by a factor 118×, MWEM+PGM by a factor 16×, PrivBayes+PGM by a factor 14.7×, RAP by a factor 47.1×, and GEM by a factor 11.7×.

2. Prior to AIM, PrivMRF was consistently the best performing mechanism, even outperforming all workload-aware mechanisms. The ALL-3WAY workload is one we expect workload agnostic mechanisms like PrivMRF to perform well on, so it is interesting, but not surprising that it outperforms workload-aware mechanisms in this setting.

3. Prior to AIM, the best *workload-aware* mechanism varied for different datasets and privacy levels: MWEM+PGM was best in 65% of settings, GEM was best

in 35% of settings [5] , and RAP was best in 0% of settings. Including AIM, we observe that it is best in 85% of settings, followed by MWEM+PGM in 11% of settings and GEM in 4% of settings. Additionally, in the most interesting regime for practical deployment ($\epsilon \geq 1.0$), AIM is best in 100% of settings.

5.6.3 TARGET workload

Results for the TARGET workload are shown in Figure 5.2. For this workload, we expect workload-aware mechanisms to have a significant advantage over workload-agnostic mechanisms, since they are aware that marginals involving the target are inherently more important for this workload. From these plots, we make the following observations:

1. All three high-level findings from the previous section are supported by these figures as well.
2. Somewhat surprisingly, PrivMRF outperforms all workload-aware mechanisms prior to AIM on this workload. This is an impressive accomplishment for PrivMRF, and clearly highlights the suboptimality of existing workload-aware mechanisms like MWEM+PGM, GEM, and RAP. Even though PrivMRF is not workload-aware, it is clear from their paper that every detail of the mechanism was carefully thought out to make the mechanism work well in practice, which explains it's impressive performance. While AIM did outperform PrivMRF again, the relative performance did not increase by a meaningful margin — offering a 1.4× improvement on average and a 4.6× improvement in the best case.

[5] We compare against a variant of GEM that selects an entire marginal query in each round. In results not shown, we also evaluated the variant of that measures a single counting query, and found that this variant performs significantly worse.

5.6.4 SKEWED workload

Results for the SKEWED workload are shown in Figure 5.3. For this workload, we again expect workload-aware mechanisms to have a significant advantage over workload-agnostic mechanisms, since they are aware of the exact (biased) set of marginals used to judge utility. From these plots, we make the following observations:

1. All four high-level findings from the previous sections are generally supported by these figures as well, with the following interesting exception:

2. PrivMRF did not score well on SALARY, and while it was still generally the second best mechanism on the other datasets (again out-performing the workload-aware mechanisms in many cases), the improvement offered by AIM over PrivMRF is much larger for this workload, averaging a $2\times$ improvement with up to a $5.7\times$ improvement in the best case. We suspect for this setting, workload-awareness is essential to achieve strong performance.

5.6.5 Tuning model capacity

In Line 12 of AIM (Algorithm 6), we construct a set of candidates to consider in the current round based on an upper limit on JT-SIZE. 80 MB was chosen to match prior work,[6] but in general we can tune it as desired to strike the right accuracy / runtime trade-off. Unlike other hyper-parameters, there is no "sweet spot" for this one: setting larger model capacities should always make the mechanism perform better, at the cost of increased runtime. We demonstrate this trade-off empirically in Figure 5.4. For $\epsilon = 0.1, 1$, and 10, we considered model capacities ranging from 1.25 MB to 1.28 GB, and ran AIM on the FIRE dataset with the ALL-3WAY workload. Results are averaged

[6]Cai et al. [16] limit the size of the *largest* clique in the junction tree to have at most 10^7 cells (80 MB with 8 byte floats), while we limit the *overall* size of the junction tree.

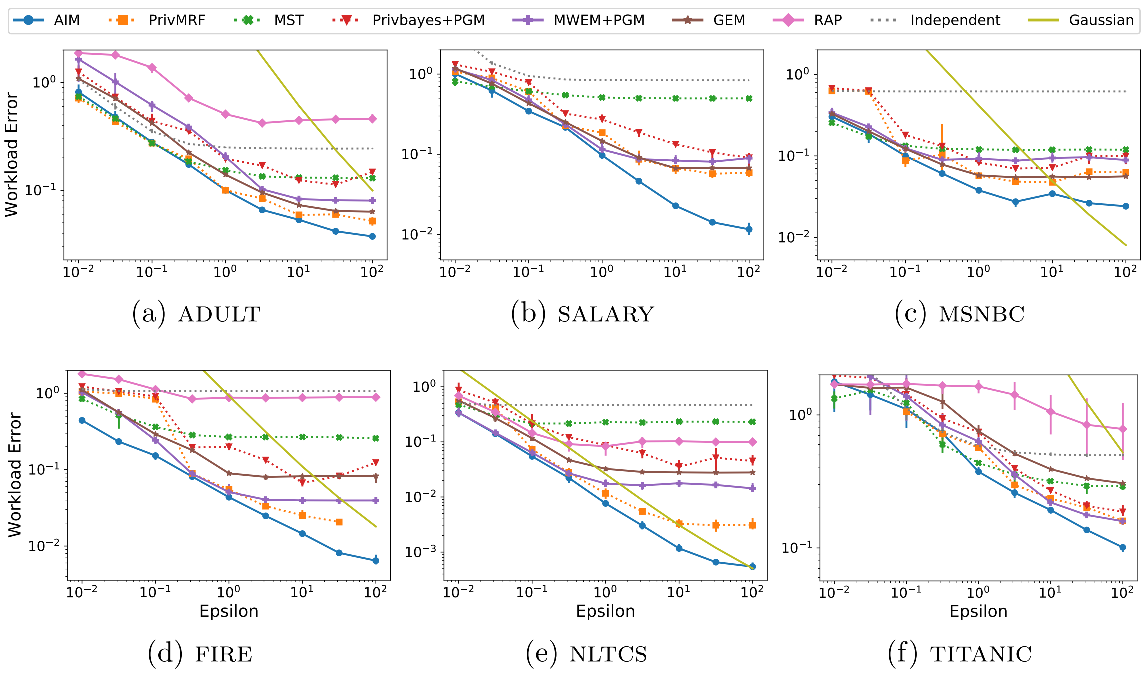


Figure 5.3: Workload error of competing mechanisms on the SKEWED workload for $\epsilon = 0.01, \ldots, 100$.

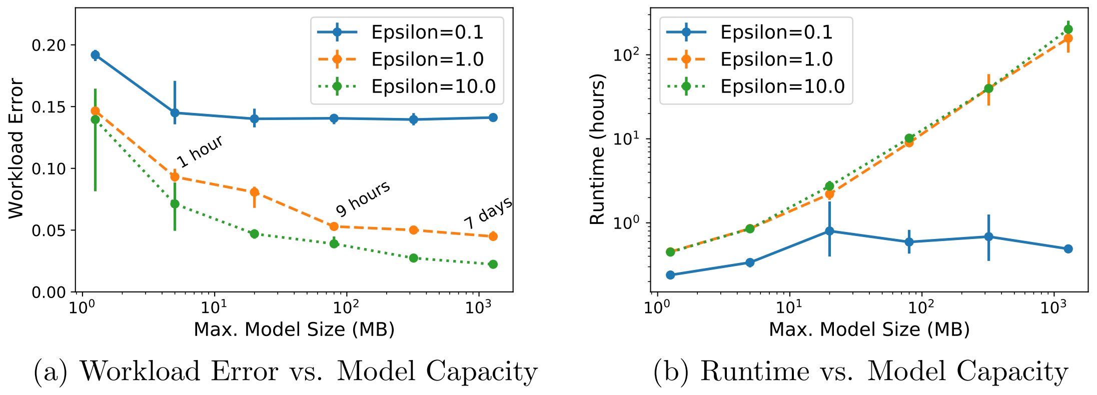


Figure 5.4: Effect of the model capacity hyperparameter on the performance of AIM.

over five trials, with error bars indicating the min/max runtime and workload error across those trials. Our main findings are listed below:

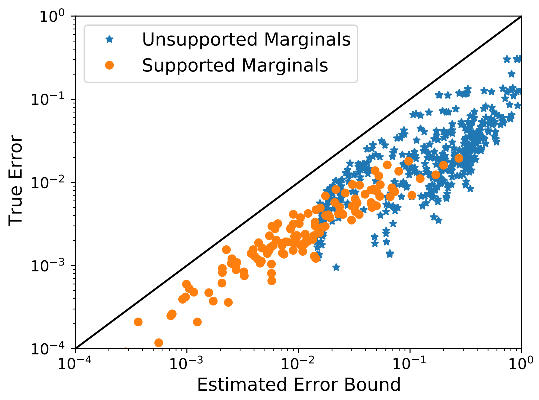


Figure 5.5: Observed error vs. 95% confidence bound on error for all cliques in the workload for the ALL-3WAY workload FIRE dataset.

1. As expected, runtime increases with model capacity, and workload error decreases with capacity. The case $\epsilon = 0.1$ is an exception, where both the plots level off beyond a capacity of 20 MB. This is because the capacity constraint is not active in this regime: AIM already favors small marginals when the available privacy budget is small by virtue of the quality score function for marginal query selection, so the model remains small even without the model capacity constraint.

2. Using the default model capacity and $\epsilon = 1$ resulted in a 9 hour runtime. We can slightly reduce error further, by about 13%, by increasing the model capacity to 1.28 GB and waiting 7 days. Conversely, we can reduce the model capacity to 5 MB which increases error by about 75%, but takes less than one hour. The law of diminishing returns is at play.

Ultimately, the model capacity to use is a policy decision. In real-world deployments, it is certainly reasonable to spend additional computational time for even a small boost in utility.

5.6.6 Uncertainty quantification

In this section, we demonstrate that our expressions for uncertainty quantification correctly bound the error, and evaluate how tight the bound is. For this experiment, we ran AIM on the FIRE dataset with the ALL-3WAY workload at $\epsilon = 10$. In Figure 5.5, we plot the true error of AIM on each marginal in the workload against the error bound predicted by our expressions. We set $\lambda = 1.7$ in Corollary 1, and $\lambda_1 = 2.7$, $\lambda_2 = 3.7$ in Corollary 2, which provides 95% confidence bounds. Our main findings are listed below:

1. For all marginals in the (downward closure of the) workload, the error bound is always greater than true error. This confirms the validity of the bound, and suggests they are safe to use in practice. Note that even if some errors were above the bounds, that would not be inconsistent with our guarantee, as at a 95% confidence level, the bound could fail to hold 5% of the time. The fact that it doesn't suggests there is some looseness in the bound.

2. The true errors and the error bounds vary considerably, ranging from 10^{-4} all the way up to and beyond 1. In general, the supported marginals have both lower errors, and lower error bounds than the unsupported marginals, which is not surprising. The error bounds are also *tighter* for the supported marginals. The median ratio between error bound and observed error is 4.4 for supported marginals and 8.3 for unsupported marginals. Intuitively, this makes sense because we know selected marginals should have higher error than non-selected marginals, but the error of the non-selected marginal can be far below that of the selected marginal (and hence the bound), which explains the larger gap between the actual error and our predicted bound.

5.7 Discussion and limitations

In this chapter, we have carefully studied the problem of workload-aware synthetic data generation under differential privacy, and presented AIM, a new mechanism for this task. AIM scales effectively to high-dimensional domains and provides state-of-the-art error rates on a variety of workloads, datasets, and privacy levels. One way AIM can immediately be improved is by modifying the model capacity parameter, whose default value is 80 MB. Increasing this value allows AIM to use a large computational budget, and it can be expected to provide better utility at the cost of increased runtime. For the default value we set, the runtime of AIM frequently exceeded 10 hours, especially at the higher values of ϵ. This could be improved by porting PrivatePGM to run on a GPU, allowing us to use AIM with larger model capacities.

CHAPTER 6
DISCUSSION AND FUTURE DIRECTIONS

In this thesis, we focused on the two related problems of linear query answering and synthetic data generation under differential privacy. To that end, we presented three new techniques to solve these problems: HDMM, PrivatePGM, and AIM. In this chapter, we briefly summarize these contributions, and discuss problems that remain open.

6.1 Summary

In Chapter 3, we presented HDMM, which overcomes the main scalability limitation of the Matrix Mechanism, and scales effectively to large multidimensional domains. For applications to which is scales, HDMM provides state-of-the-art error rates on a variety of workloads. However, HDMM still requires an explicit representation of the data vector, and therefore does not scale to high-dimensional settings where this object cannot be materialized at all.

In Chapter 4, we presented PrivatePGM, which is a general-purpose solution to the `reconstruct` sub-problem, and scales effectively to high-dimensional settings by utilizing a factored representation of the data vector. The algorithm HDMM+PGM overcomes the main scalability limitation of HDMM, although PrivatePGM is more broadly applicable outside the context of HDMM. In fact, when plugged into four state-of-the-art mechanisms, PrivatePGM was found to consistently improve their error rates across a variety of datasets.

In Chapter 5, we presented AIM, which is a workload-aware mechanism for synthetic data generation that utilizes PrivatePGM to scale to high-dimensional settings. AIM was carefully designed to extract the best ideas from prior work, while also introducing new ideas where needed to overcome their limitations. AIM offered substantial improvements over all prior work on this task in a variety of experimental settings.

6.2 Open Problems

There are a number of problems not addressed in this thesis that could be promising directions for future research, which we enumerate and discuss below.

More general workloads In this thesis, we restricted our attention to the special-but-common case of linear query workloads (Chapter 3) and marginal query workloads (Chapter 5). Extending AIM or designing new synthetic data mechanisms that work for the more general class of linear queries (perhaps defined over the low-dimensional marginals) remains an important open problem. While the prior work, MWEM+PGM, RAP, and GEM can handle workloads of this form, they achieve this by selecting a single counting query in each round, rather than a full marginal query, and thus there is likely significant room for improvement.

Beyond linear query workloads, other workloads of interest include more abstract objectives like machine learning efficacy and other non-linear query workloads. These metrics have been used to evaluate the quality of workload-agnostic synthetic data mechanisms, but have not been provided as input to the mechanisms themselves. In principle, if we know we want to run a given machine learning model on the synthetic dataset, we should be able to tailor the synthetic data to provide high utility on that model.

Mixed data types In this work, we assumed the input data was discrete, and each attribute had a finite domain with a reasonably small number of possible values. Data with numerical attributes must be appropriately discretized before using our proposed techniques, and the quality of the discretization could have a significant impact on utility of the results and the runtime of our techniques. Designing mechanisms that appropriately handle mixed (categorical and numerical) data types is an important problem. There may be more to this problem than meets the eye: a new definition of a workload and utility metric may be in order, and new types of measurements and post-processing techniques may be necessary to handle numerical data.

Public data A promising avenue for future research is to design synthetic data mechanisms that incorporate public data in a principled way. There are many places in which public data can be naturally incorporated into AIM, and exploring these ideas is a promising way to boost the utility of AIM in real world settings where public data is available. Early work on this problem includes [63, 70, 64], but this area remains under-explored.

Uncertainty quantification In Chapter 5, we developed some techniques for uncertainty quantification specific to AIM and marginal queries. In general, uncertainty quantification is an important open problem and more work is needed on it in general. Simply treating the output of a differentially private mechanism "as is" without being aware of how much error is in it can be detrimental to downstream analyses that rely on privatized data. Correctly quantifying and communicating uncertainty is crucial to avoid these misinterpretations or misuses of privatized data.

BIBLIOGRAPHY

[1] Abay, Nazmiye Ceren, Zhou, Yan, Kantarcioglu, Murat, Thuraisingham, Bhavani M., and Sweeney, Latanya. Privacy preserving synthetic data release using deep learning. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2018, Dublin, Ireland, September 10-14, 2018, Proceedings, Part I* (2018), Michele Berlingerio, Francesco Bonchi, Thomas Gärtner, Neil Hurley, and Georgiana Ifrim, Eds., vol. 11051 of *Lecture Notes in Computer Science*, Springer, pp. 510–526.

[2] Abowd, John M. The us census bureau adopts differential privacy. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2018), pp. 2867–2867.

[3] Ács, Gergely, Castelluccia, Claude, and Chen, Rui. Differentially private histogram publishing through lossy compression. In *ICDM* (2012), pp. 1–10.

[4] Asghar, Hassan Jameel, Ding, Ming, Rakotoarivelo, Thierry, Mrabet, Sirine, and Kâafar, Mohamed Ali. Differentially private release of high-dimensional datasets using the gaussian copula. *CoRR abs/1902.01499* (2019).

[5] Aydore, Sergul, Brown, William, Kearns, Michael, Kenthapadi, Krishnaram, Melis, Luca, Roth, Aaron, and Siva, Ankit A. Differentially private query release through adaptive projection. In *Proceedings of the 38th International Conference on Machine Learning* (18–24 Jul 2021), Marina Meila and Tong Zhang, Eds., vol. 139 of *Proceedings of Machine Learning Research*, PMLR, pp. 457–467.

[6] Barak, Boaz, Chaudhuri, Kamalika, Dwork, Cynthia, Kale, Satyen, McSherry, Frank, and Talwar, Kunal. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (2007), ACM, pp. 273–282.

[7] Beck, Amir, and Teboulle, Marc. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters 31*, 3 (2003), 167–175.

[8] Bernstein, Garrett, McKenna, Ryan, Sun, Tao, Sheldon, Daniel, Hay, Michael, and Miklau, Gerome. Differentially private learning of undirected graphical models using collective graphical models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 478–487.

[9] Bhaskara, Aditya, Dadush, Daniel, Krishnaswamy, Ravishankar, and Talwar, Kunal. Unconditional differentially private mechanisms for linear queries. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 2012), STOC '12, ACM, pp. 1269–1284.

[10] Bindschaedler, Vincent, Shokri, Reza, and Gunter, Carl A. Plausible deniability for privacy-preserving data synthesis. *Proceedings of the VLDB Endowment 10*, 5 (2017), 481–492.

[11] Błasiok, Jaroslaw, Bun, Mark, Nikolov, Aleksandar, and Steinke, Thomas. Towards instance-optimal private query release. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms* (2019), SIAM, pp. 2480–2497.

[12] Blum, Avrim, Ligett, Katrina, and Roth, Aaron. A learning theory approach to noninteractive database privacy. *Journal of the ACM (JACM) 60*, 2 (2013), 1–25.

[13] Bun, Mark, and Steinke, Thomas. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference* (2016), Springer, pp. 635–658.

[14] Bun, Mark, Ullman, Jonathan, and Vadhan, Salil. Fingerprinting codes and the price of approximate differential privacy. *SIAM Journal on Computing 47*, 5 (2018), 1888–1938.

[15] Cadez, Igor, Heckerman, David, Meek, Christopher, Smyth, Padhraic, and White, Steven. Visualization of navigation patterns on a web site using model-based clustering. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining* (2000), pp. 280–284.

[16] Cai, Kuntai, Lei, Xiaoyu, Wei, Jianxin, and Xiao, Xiaokui. Data synthesis via differentially private markov random fields. *Proceedings of the VLDB Endowment 14*, 11 (2021), 2190–2202.

[17] Canonne, Clément L., Kamath, Gautam, and Steinke, Thomas. The discrete gaussian for differential privacy. In *NeurIPS* (2020).

[18] Cesar, Mark, and Rogers, Ryan. Bounding, concentrating, and truncating: Unifying privacy loss composition for data analytics. In *Proceedings of the 32nd International Conference on Algorithmic Learning Theory* (16–19 Mar 2021), Vitaly Feldman, Katrina Ligett, and Sivan Sabato, Eds., vol. 132 of *Proceedings of Machine Learning Research*, PMLR, pp. 421–457.

[19] Charest, Anne-Sophie. How can we analyze differentially-private synthetic datasets? *Journal of Privacy and Confidentiality 2*, 2 (2011).

[20] Chen, Rui, Xiao, Qian, Zhang, Yu, and Xu, Jianliang. Differentially private high-dimensional data publication via sampling-based inference. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2015), ACM, pp. 129–138.

[21] Chen, Tianqi, He, Tong, Benesty, Michael, Khotilovich, Vadim, Tang, Yuan, Cho, Hyunsu, Chen, Kailong, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2 1*, 4 (2015), 1–4.

[22] Cormode, Graham, Procopiuc, Cecilia M., Srivastava, Divesh, Shen, Entong, and Yu, Ting. Differentially private spatial decompositions. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012* (2012), Anastasios Kementsietsidis and Marcos Antonio Vaz Salles, Eds., IEEE Computer Society, pp. 20–31.

[23] Desfontains, Damien, 2022.

[24] Devaux, Elise. List of synthetic data startups and companies — 2021.

[25] Ding, Bolin, Kulkarni, Janardhan, and Yekhanin, Sergey. Collecting telemetry data privately. In *Advances in Neural Information Processing Systems* (2017), pp. 3571–3580.

[26] Ding, Bolin, Winslett, Marianne, Han, Jiawei, and Li, Zhenhui. Differentially private data cubes: optimizing noise sources and consistency. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011* (2011), Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis, Eds., ACM, pp. 217–228.

[27] Dinur, Irit, and Nissim, Kobbi. Revealing information while preserving privacy. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA* (2003), Frank Neven, Catriel Beeri, and Tova Milo, Eds., ACM, pp. 202–210.

[28] Dwork, Cynthia, Naor, Moni, Reingold, Omer, and Rothblum, Guy N. Pure differential privacy for rectangle queries via private partitions. In *International Conference on the Theory and Application of Cryptology and Information Security* (2015), Springer, pp. 735–751.

[29] Dwork, Cynthia, Nikolov, Aleksandar, and Talwar, Kunal. Efficient algorithms for privately releasing marginals via convex relaxations. *Discrete & Computational Geometry 53*, 3 (2015), 650–673.

[30] Dwork, Cynthia, Nissim, Frank McSherry Kobbi, and Smith, Adam. Calibrating noise to sensitivity in private data analysis. In *TCC* (2006), pp. 265–284.

[31] Dwork, Cynthia, and Roth, Aaron. *The Algorithmic Foundations of Differential Privacy.* Found. and Trends in Theoretical Computer Science, 2014.

[32] Dwork, Cynthia, Rothblum, Guy N, and Vadhan, Salil. Boosting and differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science* (2010), IEEE, pp. 51–60.

[33] Erlingsson, Úlfar, Pihur, Vasyl, and Korolova, Aleksandra. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security* (2014), ACM, pp. 1054–1067.

[34] Fong, David Chin-Lung, and Saunders, Michael. Lsmr: An iterative algorithm for sparse least-squares problems. *SIAM Journal on Scientific Computing 33*, 5 (2011), 2950–2971.

[35] Frank E. Harrell Jr., Thomas Cason. Encyclopedia titanica.

[36] Gaboardi, Marco, Arias, Emilio Jesús Gallego, Hsu, Justin, Roth, Aaron, and Wu, Zhiwei Steven. Dual query: Practical private query release for high dimensional data. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014* (2014), vol. 32 of *JMLR Workshop and Conference Proceedings*, JMLR.org, pp. 1170–1178.

[37] Gaboardi, Marco, Arias, Emilio Jesús Gallego, Hsu, Justin, Roth, Aaron, and Wu, Zhiwei Steven. Dual query: Practical private query release for high dimensional data. In *Proceedings of the 31st International Conference on Machine Learning (ICML)* (2014).

[38] Ge, Chang, Mohapatra, Shubhankar, He, Xi, and Ilyas, Ihab F. Kamino: Constraint-aware differentially private data synthesis. *Proceedings of the VLDB Endowment 14*, 10 (2021), 1886–1899.

[39] Greene, William H. *Econometric analysis*. Pearson Education India, 2003.

[40] Gupta, Anupam, Roth, Aaron, and Ullman, Jonathan. Iterative constructions and private data release. In *Theory of cryptography conference* (2012), Springer, pp. 339–356.

[41] Hardt, Moritz, Ligett, Katrina, and McSherry, Frank. A simple and practical algorithm for differentially private data release. In *Advances in Neural Information Processing Systems* (2012), pp. 2339–2347.

[42] Hardt, Moritz, Ligett, Katrina, and McSherry, Frank. A simple and practical algorithm for differentially private data release. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States* (2012), Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, Eds., pp. 2348–2356.

[43] Hardt, Moritz, and Rothblum, Guy N. A multiplicative weights mechanism for privacy-preserving data analysis. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on* (2010), IEEE, pp. 61–70.

[44] Hardt, Moritz, and Talwar, Kunal. On the geometry of differential privacy. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing* (New York, NY, USA, 2010), STOC '10, ACM, pp. 705–714.

[45] Hartung, Joachim, Knapp, Guido, Sinha, Bimal K, and Sinha, Bimal K. *Statistical meta-analysis with applications*, vol. 6. Wiley Online Library, 2008.

[46] Hay, Michael, Machanavajjhala, Ashwin, Miklau, Gerome, Chen, Yan, and Zhang, Dan. Principled evaluation of differentially private algorithms using dpbench. In *Proceedings of the 2016 International Conference on Management of Data* (2016), pp. 139–154.

[47] Hay, Michael, Rastogi, Vibhor, Miklau, Gerome, and Suciu, Dan. Boosting the accuracy of differentially private histograms through consistency. *PVLDB 3*, 1-2 (2010), 1021–1032.

[48] Huang, Zhiqi, McKenna, Ryan, Bissias, George, Miklau, Gerome, Hay, Michael, and Machanavajjhala, Ashwin. Psyndb: accurate and accessible private data generation. *VLDB Demo*.

[49] Johnson, Noah, Near, Joseph P, and Song, Dawn. Towards practical differential privacy for sql queries. *Proceedings of the VLDB Endowment 11*, 5 (2018), 526–539.

[50] Jordon, James, Yoon, Jinsung, and van der Schaar, Mihaela. PATE-GAN: generating synthetic data with differential privacy guarantees. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019* (2019), OpenReview.net.

[51] Ke, Guolin, Meng, Qi, Finley, Thomas, Wang, Taifeng, Chen, Wei, Ma, Weidong, Ye, Qiwei, and Liu, Tie-Yan. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems 30* (2017).

[52] Kohavi, Ron, et al. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Kdd* (1996), vol. 96, pp. 202–207.

[53] Koller, Daphne, and Friedman, Nir. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[54] Lancaster, Peter, and Farahat, Hanafi K. Norms on direct sums and tensor products. *mathematics of computation 26*, 118 (1972), 401–414.

[55] Lee, Jaewoo, Wang, Yue, and Kifer, Daniel. Maximum likelihood postprocessing for differential privacy under consistency constraints. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2015), ACM, pp. 635–644.

[56] Li, Chao, Hay, Michael, Miklau, Gerome, and Wang, Yue. A data-and workload-aware algorithm for range queries under differential privacy. *PVLDB 7*, 5 (2014), 341–352.

[57] Li, Chao, Hay, Michael, Rastogi, Vibhor, Miklau, Gerome, and McGregor, Andrew. Optimizing linear counting queries under differential privacy. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (2010), ACM, pp. 123–134.

[58] Li, Chao, and Miklau, Gerome. An adaptive mechanism for accurate query answering under differential privacy. *PVLDB 5*, 6 (2012), 514–525.

[59] Li, Chao, and Miklau, Gerome. Optimal error of query sets under the differentially-private matrix mechanism. In *ICDT* (2013).

[60] Li, Chao, Miklau, Gerome, Hay, Michael, McGregor, Andrew, and Rastogi, Vibhor. The matrix mechanism: optimizing linear counting queries under differential privacy. *The VLDB Journal 24*, 6 (2015), 757–781.

[61] Li, Haoran, Xiong, Li, and Jiang, Xiaoqian. Differentially private synthesization of multi-dimensional data using copula functions. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014* (2014), Sihem Amer-Yahia, Vassilis Christophides, Anastasios Kementsietsidis, Minos N. Garofalakis, Stratos Idreos, and Vincent Leroy, Eds., OpenProceedings.org, pp. 475–486.

[62] Liu, Fang. Model-based differentially private data synthesis. *arXiv preprint arXiv:1606.08052* (2016).

[63] Liu, Terrance, Vietri, Giuseppe, Steinke, Thomas, Ullman, Jonathan R., and Wu, Zhiwei Steven. Leveraging public data for practical private query release. In *ICML* (2021), pp. 6968–6977.

[64] Liu, Terrance, Vietri, Giuseppe, and Wu, Steven. Iterative methods for private synthetic data: Unifying framework and new methods. In *Advances in Neural Information Processing Systems* (2021), A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, Eds.

[65] Machanavajjhala, Ashwin, Kifer, Daniel, Abowd, John, Gehrke, Johannes, and Vilhuber, Lars. Privacy: Theory meets practice on the map. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering* (2008), IEEE Computer Society, pp. 277–286.

[66] Manton, Kenneth G. National long-term care survey: 1982, 1984, 1989, 1994, 1999, and 2004, 2010.

[67] McKenna, Ryan. Workloads of Counting Queries: Enabling Rich Statistical Analyses with Differential Privacy . `https://www.nist.gov/blogs/cybersecurity-insights/workloads-counting-queries-enabling-rich-statistical-analyses`, 2021. [Online; accessed 05-March-2022].

[68] McKenna, Ryan, and Liu, Terrance. A simple recipe for private synthetic data generation, Jan 2022.

[69] McKenna, Ryan, Miklau, Gerome, Hay, Michael, and Machanavajjhala, Ashwin. Optimizing error of high-dimensional statistical queries under differential privacy. *Proceedings of the VLDB Endowment 11*, 10 (2018), 1206–1219.

[70] McKenna, Ryan, Miklau, Gerome, and Sheldon, Daniel. Winning the nist contest: A scalable and general approach to differentially private synthetic data. *Journal of Privacy and Confidentiality 11*, 3 (2021).

[71] McKenna, Ryan, Mullins, Brett, Sheldon, Daniel, and Miklau, Gerome. Aim: An adaptive and iterative mechanism for differentially private synthetic data.

[72] McKenna, Ryan, Pradhan, Siddhant, Sheldon, Daniel R, and Miklau, Gerome. Relaxed marginal consistency for differentially private query answering. *Advances in Neural Information Processing Systems 34* (2021).

[73] McKenna, Ryan, Sheldon, Daniel, and Miklau, Gerome. Graphical-model based estimation and inference for differential privacy. In *International Conference on Machine Learning* (2019), pp. 4435–4444.

[74] McKenna, Ryan, Wu, Joie, Tajima, Arisa, Mullins, Brett, Ferrando, Cecilia, and Pradhan, Siddhant. Adaptive Granularity Mechanism, 10 2021.

[75] McSherry, Frank, and Talwar, Kunal. Mechanism design via differential privacy. In *FOCS* (2007).

[76] Messing, Solomon, DeGregorio, Christina, Hillenbrand, Bennett, King, Gary, Mahanti, Saurav, Mukerjee, Zagreb, Nayak, Chaya, Persily, Nate, State, Bogdan, and Wilkins, Arjun. Facebook Privacy-Protected Full URLs Data Set, 2020.

[77] Murphy, Kevin P. *Machine learning: a probabilistic perspective.* MIT press, 2012.

[78] Nesterov, Yurii. Primal-dual subgradient methods for convex problems. *Mathematical programming 120*, 1 (2009), 221–259.

[79] Nikolov, Aleksandar. An improved private mechanism for small databases. In *International Colloquium on Automata, Languages, and Programming* (2015), Springer, pp. 1010–1021.

[80] Nikolov, Aleksandar, Talwar, Kunal, and Zhang, Li. The geometry of differential privacy: the sparse and approximate cases. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing* (2013), pp. 351–360.

[81] Parikh, Neal, Boyd, Stephen, et al. Proximal algorithms. *Foundations and Trends® in Optimization 1*, 3 (2014), 127–239.

[82] Qardaji, Wahbeh, Yang, Weining, and Li, Ninghui. Differentially private grids for geospatial data. In *Intl. Conference on Data Engineering (ICDE)* (2013), IEEE, pp. 757–768.

[83] Qardaji, Wahbeh, Yang, Weining, and Li, Ninghui. Understanding hierarchical methods for differentially private histograms. *PVLDB 6*, 14 (2013), 1954–1965.

[84] Qardaji, Wahbeh H., Yang, Weining, and Li, Ninghui. Priview: practical differentially private release of marginal contingency tables. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014* (2014), Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu, Eds., ACM, pp. 1435–1446.

[85] Raskhodnikova, Sofya, and Smith, Adam. Lipschitz extensions for node-private graph statistics and the generalized exponential mechanism. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)* (2016), IEEE, pp. 495–504.

[86] Ridgeway, Diane, Theofanos, Mary, Manley, Terese, Task, Christine, et al. Challenge design and lessons learned from the 2018 differential privacy challenges.

[87] Rogers, Ryan, Subramaniam, Subbu, Peng, Sean, Durfee, David, Lee, Seunghyun, Kancha, Santosh Kumar, Sahay, Shraddha, and Ahammad, Parvez. Linkedin's audience engagements api: A privacy preserving data analytics system at scale. *arXiv preprint arXiv:2002.05839* (2020).

[88] Rogers, Ryan M, Roth, Aaron, Ullman, Jonathan, and Vadhan, Salil. Privacy odometers and filters: Pay-as-you-go composition. *Advances in Neural Information Processing Systems 29* (2016), 1921–1929.

[89] Tantipongpipat, Uthaipon, Waites, Chris, Boob, Digvijay, Siva, Amaresh Ankit, and Cummings, Rachel. Differentially private mixed-type data generation for unsupervised learning. *CoRR abs/1912.03250* (2019).

[90] Thakurta, Abhradeep Guha, Vyrros, Andrew H, Vaishampayan, Umesh S, Kapoor, Gaurav, Freudinger, Julien, Prakash, Vipul Ved, Legendre, Arnaud, and Duplinsky, Steven. Emoji frequency detection and deep link frequency, July 11 2017. US Patent 9,705,908.

[91] Torfi, Amirsina, Fox, Edward A, and Reddy, Chandan K. Differentially private synthetic medical data generation using convolutional gans. *Information Sciences 586* (2022), 485–500.

[92] Torkzadehmahani, Reihaneh, Kairouz, Peter, and Paten, Benedict. DP-CGAN: differentially private synthetic data and label generation. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019, Long Beach, CA, USA, June 16-20, 2019* (2019), Computer Vision Foundation / IEEE, pp. 98–104.

[93] Tsagris, Michail, Beneki, Christina, and Hassani, Hossein. On the folded normal distribution. *Mathematics 2*, 1 (2014), 12–28.

[94] Van Loan, Charles F. The ubiquitous kronecker product. *Journal of computational and applied mathematics 123*, 1 (2000), 85–100.

[95] Vietri, Giuseppe, Tian, Grace, Bun, Mark, Steinke, Thomas, and Wu, Zhiwei Steven. New oracle-efficient algorithms for private synthetic data release. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event* (2020), vol. 119 of *Proceedings of Machine Learning Research*, PMLR, pp. 9765–9774.

[96] Vilnis, Luke, Belanger, David, Sheldon, Daniel, and McCallum, Andrew. Bethe projections for non-local inference. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence* (2015), AUAI Press, pp. 892–901.

[97] Wainwright, Martin J., and Jordan, Michael I. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning 1*, 1-2 (2008), 1–305.

[98] www.nist.gov. 2018 differential privacy synthetic data challenge. `https://www.nist.gov/ctl/pscr/open-innovation-prize-challenges/past-prize-challenges/2018-differential-privacy-synthetic`, 2018.

[99] Xiao, Lin. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research 11*, Oct (2010), 2543–2596.

[100] Xiao, Xiaokui, Wang, Guozhang, and Gehrke, Johannes. Differential privacy via wavelet transforms. *IEEE Transactions on Knowledge and Data Engineering 23*, 8 (2011), 1200–1214.

[101] Xiao, Yonghui, Xiong, Li, Fan, Liyue, Goryczka, Slawomir, and Li, Haoran. DPCube: Differentially private histogram release through multidimensional partitioning. *Transactions of Data Privacy 7*, 3 (2014).

[102] Xie, Liyang, Lin, Kaixiang, Wang, Shu, Wang, Fei, and Zhou, Jiayu. Differentially private generative adversarial network. *CoRR abs/1802.06739* (2018).

[103] Xu, Chugui, Ren, Ju, Zhang, Yaoxue, Qin, Zhan, and Ren, Kui. Dppro: Differentially private high-dimensional data release via random projection. *IEEE Transactions on Information Forensics and Security 12*, 12 (2017), 3081–3093.

[104] Xu, Jia, Zhang, Zhenjie, Xiao, Xiaokui, Yang, Yin, and Yu, Ge. Differentially private histogram publication. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on* (2012), pp. 32–43.

[105] Xu, Jia, Zhang, Zhenjie, Xiao, Xiaokui, Yang, Yin, Yu, Ge, and Winslett, Marianne. Differentially private histogram publication. *The VLDB Journal* (2013), 1–26.

[106] Yaroslavtsev, Grigory, Cormode, Graham, Procopiuc, Cecilia M., and Srivastava, Divesh. Accurate and efficient private release of datacubes and contingency tables. In *ICDE* (2013).

[107] Yuan, Ganzhao, Yang, Yin, Zhang, Zhenjie, and Hao, Zhifeng. Convex optimization for linear query processing under approximate differential privacy. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), ACM, pp. 2005–2014.

[108] Yuan, Ganzhao, Zhang, Zhenjie, Winslett, Marianne, Xiao, Xiaokui, Yang, Yin, and Hao, Zhifeng. Low-rank mechanism: optimizing batch queries under differential privacy. *PVLDB 5*, 11 (2012), 1352–1363.

[109] Zhang, Dan, McKenna, Ryan, Kotsogiannis, Ios, Hay, Michael, Machanavajjhala, Ashwin, and Miklau, Gerome. Ektelo: A framework for defining differentially-private computations. In *Conference on Management of Data (SIGMOD)* (2018).

[110] Zhang, Jun, Cormode, Graham, Procopiuc, Cecilia M., Srivastava, Divesh, and Xiao, Xiaokui. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems (TODS) 42*, 4 (2017), 25:1–25:41.

[111] Zhang, Jun, Cormode, Graham, Procopiuc, Cecilia M, Srivastava, Divesh, and Xiao, Xiaokui. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems (TODS) 42*, 4 (2017), 25.

[112] Zhang, Jun, Xiao, Xioakui, and Xie, Xing. Privtree: A differentially private algorithm for hierarchical decompositions. In *SIGMOD* (2016).

[113] Zhang, Wei, Zhao, Jingwen, Wei, Fengqiong, and Chen, Yunfang. Differentially private high-dimensional data publication via markov network. *EAI Endorsed Trans. Security Safety 6*, 19 (2019), e4.

[114] Zhang, Xiaojian, Chen, Rui, Xu, Jianliang, Meng, Xiaofeng, and Xie, Yingtao. Towards accurate histogram publication under differential privacy. In *SDM* (2014).

[115] Zhang, Xinyang, Ji, Shouling, and Wang, Ting. Differentially private releasing via deep generative model (technical report). *arXiv preprint arXiv:1801.01594* (2018).

[116] Zhang, Zhikun, Wang, Tianhao, Li, Ninghui, Honorio, Jean, Backes, Michael, He, Shibo, Chen, Jiming, and Zhang, Yang. Privsyn: Differentially private data synthesis. In *30th USENIX Security Symposium (USENIX Security 21)* (Aug. 2021), USENIX Association, pp. 929–946.