Optimizing error of high-dimensional statistical queries under differential privacy*

Ryan McKenna†, Gerome Miklau†,*, Michael Hay‡,*, Ashwin Machanavajjhala**
† Univ. of Massachusetts, Amherst, College of Information and Computing Sciences
‡ Colgate University, Dept. of Computer Science
** Duke University, Dept. of Computer Science
* U.S. Census Bureau, Suitland, MD
{rmckenna, miklau}@cs.umass.edu mhay@colgate.edu ashwin@cs.duke.edu

ABSTRACT

Differentially private algorithms for answering sets of predicate counting queries on a sensitive database have many applications. Organizations that collect individual-level data, such as statistical agencies and medical institutions, use them to safely release summary tabulations. However, existing techniques are accurate only on a narrow class of query workloads, or are extremely slow, especially when analyzing more than one or two dimensions of the data.

In this work we propose HDMM, a new differentially private algorithm for answering a workload of predicate counting queries, that is especially effective for higher-dimensional datasets. HDMM represents query workloads using an implicit matrix representation and exploits this compact representation to efficiently search (a subset of) the space of differentially private algorithms for one that answers the input query workload with high accuracy. We empirically show that HDMM can efficiently answer queries with lower error than state-of-the-art techniques on a variety of low and high dimensional datasets.

PVLDB Reference Format:
Ryan McKenna, Gerome Miklau, Michael Hay, Ashwin Machanavajjhala. Optimizing error of high-dimensional statistical queries under differential privacy. *PVLDB*, 11 (10): 1206-1219, 2018.
DOI: https://doi.org/10.14778/3231751.3231769

1. INTRODUCTION

Institutions like the U.S. Census Bureau and Medicare regularly release summary statistics about individuals, including population statistics cross-tabulated by demographic attributes [6,32] and tables reporting on hospital discharges organized by medical condition and patient characteristics [20]. These data have the potential to reveal sensitive information, especially through joint analysis of multiple releases [16,29,36]. Differential privacy [11,12] has become the dominant standard for ensuring the privacy of such data releases. An algorithm for releasing statistics over a dataset satisfies ϵ-differential privacy if adding or removing a single record in the input dataset does not result in a significant change in the output of the algorithm. The allowable change is determined by ϵ, also called the privacy-loss budget. If each record in the input corresponds to a unique individual, this notion gives a compelling privacy guarantee [23].

We consider the problem of releasing answers to a *workload* (i.e., a set) of *predicate counting queries* while satisfying ϵ-differential privacy. Predicate counting queries have the form `SELECT Count(*) FROM R WHERE` ϕ, where ϕ is any boolean formula over the attributes in `R`. (This problem formulation also supports group-by queries, each of which can be rewritten into a set of predicate counting queries, one query per possible group.) Workloads of such queries are quite versatile, expressing histograms, multi-dimensional range queries, data cubes, marginals, or arbitrary combinations thereof.

There has been a plethora of work on differentially private techniques for answering sets of queries including work establishing theoretical lower bounds [4,17,31] and practical algorithms [2,3,8,10,19,24–26,28,33–35,35,41–50].

One class of techniques answers the queries of interest on the database and then uses the Laplace Mechanism to add noise, calibrated to their *sensitivity*, or the maximum change in answers resulting from one change in the input database [11,13,21,40]. These techniques can answer queries using off-the-shelf systems (queries in SQL and data in relational form), and thus can be implemented efficiently [22,30]. However, a key limitation of this class is that because the noise is calibrated on a per-query basis, they fail to exploit workload structure and thus add more noise than is strictly necessary, resulting in suboptimal accuracy.

A second, more sophisticated, approach to query answering generalizes the Laplace Mechanism by first **selecting** a new set of *strategy* queries, then **measuring** the strategy queries using the Laplace mechanism, and **reconstructing** answers to the input queries from the noisy measurements. Choosing an effective query answering strategy (different from the workload) can result in orders-of-magnitude lower error than the Laplace mechanism, with no cost to privacy.

An example of a technique from the select-measure-reconstruct paradigm is the Matrix Mechanism (MM) [28], illustrated in Table 1a. The MM, and other techniques in this paradigm, represent the database and queries in vector form, expressed over the full domain of each tuple (the

*Views expressed in this paper are those of authors and do not necessarily reflect the views of the U.S. Census Bureau.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were invited to present their results at The 44th International Conference on Very Large Data Bases, August 2018, Rio de Janeiro, Brazil.
Proceedings of the VLDB Endowment, Vol. 11, No. 10
Copyright 2018 VLDB Endowment 2150-8097/18/06.
DOI: https://doi.org/10.14778/3231751.3231769

Table 1: Overview of the *High Dimensional Matrix Mechanism* (HDMM), compared with the Matrix Mechanism (MM) [25].

(a) The Matrix Mechanism (MM) [25]

Input: workload $\boldsymbol{W}$, in matrix form
data $\boldsymbol{x}$, in vector form
privacy parameter ϵ

SELECT	$\boldsymbol{A}$	=	$\mathsf{OPT}_{MM}(\boldsymbol{W})$
MEASURE	$\boldsymbol{a}$	=	$\boldsymbol{A}\boldsymbol{x}$
	$\boldsymbol{y}$	=	$\boldsymbol{a} + Lap(\|\|\boldsymbol{A}\|\|_1/\epsilon)$
RECONSTRUCT	$\bar{\boldsymbol{x}}$	=	$\boldsymbol{A}^+\boldsymbol{y}$
	ans	=	$\boldsymbol{W}\bar{\boldsymbol{x}}$

(b) HDMM Overview

Input: workload $\mathcal{W}$, in logical form
data $\boldsymbol{x}$, in vector form
privacy parameter ϵ

$\mathbb{W}$	=	$\mathsf{ImpVec}(\mathcal{W})$	*// Compact vector representation*
$\mathbb{A}$	=	$\mathsf{OPT}_{\mathsf{HDMM}}(\mathbb{W})$	*// Optimized strategy selection*
$\boldsymbol{a}$	=	$\mathsf{Multiply}(\mathbb{A}, \boldsymbol{x})$	*// Strategy query answering*
$\boldsymbol{y}$	=	$\boldsymbol{a} + Lap(\|\|\mathbb{A}\|\|_1/\epsilon)$	*// Noise addition*
$\bar{\boldsymbol{x}}$	=	$\mathsf{LstSqr}(\mathbb{A}, \boldsymbol{y})$	*// Inference*
ans	=	$\mathsf{Multiply}(\mathbb{W}, \bar{\boldsymbol{x}})$	*// Workload answering*

product of the domains of the attributes). The vector representation allows these techniques to compute the sensitivity of sets of queries using a matrix norm, and to use inference algorithms based on linear algebra to reconstruct answers from noisy measurements. In this vector form, the selection step corresponds to selecting a query matrix $\boldsymbol{A}$ (the strategy), and the measurement step reduces to computing the matrix-vector product between $\boldsymbol{A}$ and the data vector $\boldsymbol{x}$.

Many recent algorithms fall within the select-measure-reconstruct paradigm [2, 8, 10, 19, 24–26, 28, 28, 33–35, 41–47, 49,50], differing primarily in the measurement selection step. We can characterize measurement selection as a search problem over a space of strategies, distinguishing prior work in terms of key algorithmic design choices: the search space, the cost function, and the type of search algorithm (greedy, local, global, etc.). These design choices impact the three key performance considerations: accuracy, runtime, and scalability (in terms of increasing dataset dimensionality).

At one extreme are techniques that explore a narrow search space, making them efficient and scalable but not particularly accurate (in particular, their search space may include accurate strategies only for a limited class of workloads). For example, HB [34] considers strategies consisting of hierarchically structured interval queries. It performs a simple search to find the branching factor of the hierarchical strategy that minimizes an error measure that assumes the workload consists of all range queries (regardless of the actual input workload). It is efficient and can scale to higher dimensions, but it achieves competitive accuracy only when the workload consists of range queries and the data is low dimensional.

At the other extreme are techniques that search a large space, and adapt to the workload by finding a strategy within that space that offers low error on the workload, thereby making them capable of producing a more accurate strategy for the particular input. However, this increased accuracy comes at the cost of high runtime and poor scalability. This is exemplified by MM, which solves a rank-constrained semi-definite program to find the *optimal* solution. Unfortunately, the optimization program is infeasible to execute on any non-trivial input workload.

In short, there is no prior work that is accurate for a wide range of input workloads, sufficiently fast, and capable of scaling to large multi-dimensional domains.

Overview of approach and contributions. We describe the High-Dimensional Matrix Mechanism (HDMM), a new algorithm for answering workloads of predicate counting queries. While similar in spirit to the matrix mechanism, there are a number of innovations that make it more efficient and scalable. We contrast the two algorithms in Table 1.

First, MM represents query workloads as fully-materialized matrices, while HDMM uses a compact *implicit* matrix representation of the logical queries, which we call a *union of products* (Section 4), for which query sizes are not exponential in the number of attributes. In the use case we will describe soon, the matrix representation of one of the workloads would be 22TB; in contrast, our most compact representation of this workload is just 687KB. Without this innovation it is infeasible merely to evaluate the error of a strategy, let alone select the one with the least error.

The second key difference between the matrix mechanism and HDMM is the search algorithm underlying the SELECT step, and it is a key technical innovation of this paper. HDMM uses a set of optimization routines (described in Sections 5 and 6) that can exploit our compact implicit workload representation. These different optimization routines work by restricting search to different regions of the strategy space: local optimization is tractable in these regions but they still contain high quality strategies. The output is a measurement strategy $\mathbb{A}$, also represented in a compact implicit form.

Our third innovation consists of efficient techniques for measurement and reconstruction. In MM, these steps are implemented by multiplying a matrix $\boldsymbol{A}$ with the data vector $\boldsymbol{x}$ and multiplying a matrix pseudo-inverse $\boldsymbol{A}^+$ with the noisy answers $\boldsymbol{y}$, respectively. The latter inference step can be inefficient in explicit matrix form. HDMM exploits the special structure of our selected measurements to speed up these steps, as described in Section 7.2.

As a result of these innovations, HDMM achieves high accuracy on a *variety* of realistic input workloads, in both low and high dimensions. In fact, in our experiments, we find it has higher accuracy than all prior select-measure-reconstruct techniques, even on inputs for which the prior techniques were specifically designed (e.g., it is more accurate than HB on range queries). We also find it is more accurate than state-of-the-art techniques outside the select-measure-reconstruct paradigm. It achieves reasonable runtime and scales more effectively than prior work that performs non-trivial optimization (see Section 8 for a detailed scalability evaluation).

Organization. In addition to the sections noted above, we describe our use case next, followed by background, and the end-to-end algorithm components in Section 7, experiments in Section 8, and discussion in Section 9.

2. MOTIVATING USE CASE

Based on our collaboration with the U.S. Census Bureau[1], we use as a running example and motivating use case the differentially private release of a collection of 10 tabulations from the 2010 *Summary File 1 (SF1)* [6], an important data product based on the Census of Population and Housing (CPH). Statistics from SF1 are used for redistricting, demographic projections, and other policy-making.

Our workload is a subset of queries from SF1 that can be written as predicate counting queries over a `Person` relation. (We omit other queries involving households; for brevity we refer to our selected queries as simply SF1.) The `Person` relation has the following schema: six boolean attributes describing Race, two boolean attributes for Hispanic Ethnicity and Sex, Age in years between 0 and 114, and a Relationship-to-householder field that has 17 values. Our SF1 workload has 4151 predicate counting queries, each of the form `SELECT Count(*) FROM Person WHERE` ϕ, where ϕ specifies some combination of demographic properties (e.g. number of Persons who are Male, over 18, and Hispanic) and thus each query reports a count at the national level. These queries are on a multidimensional domain of size $2^6 \times 2 \times 2 \times 115 \times 17 = 500{,}480$. The data also includes a geographic attribute encoding state (51 values including D.C.). A workload we call SF1+ consists of the national level queries in SF1 *as well as* the same queries at the state level for each of 51 states. We can succinctly express the state level queries as an additional 4151 queries of the form: `SELECT state, Count(*) FROM Person WHERE` ϕ `GROUP BY state`. Thus, SF1+ can be represented by a total of $4151 + 4151 = 8302$ SQL queries. The SF1+ queries are defined on a domain of size $500{,}480 \times 51 = 25{,}524{,}480$.

In addition to their SQL representation, the SF1 and SF1+ workloads can be naturally expressed in a logical form defined in Section 4.1. We use $\mathcal{W}_{\text{SF1}}$and $\mathcal{W}_{\text{SF1+}}$to denote the logical forms of SF1 and SF1+ respectively.

3. BACKGROUND

We describe below the relevant background, including the data model, logical query workloads, their corresponding vector representations and differential privacy.

3.1 Data and schema

We assume a single-table relational schema $R(A_1 \dots A_d)$, where $attr(R)$ denotes the set of attributes of R. Subsets of attributes are denoted $\mathcal{A} \subseteq attr(R)$. Each attribute A_i has a finite domain $dom(A_i)$. The full domain of R is $dom(R) = dom(A_1) \times \cdots \times dom(A_d)$, containing all possible tuples conforming to R. An instance I of relation R is a multiset whose elements are tuples in $dom(R)$. We use N for $|dom(R)|$.

[1]The Census Bureau recently announced [7] that the test publications produced by the 2018 End-to-End Census Test would be protected by a disclosure limitation system based on differential privacy. The End-to-End Test is a prototype of the full production system to be used for the 2020 Census of Population and Housing. If the test of this disclosure limitation system is successful, then the expectation is that the publications of the 2020 Census will also be protected using differential privacy. The work discussed in this paper is part of the research and development activity for those disclosure limitation systems.

3.2 Logical view of queries

Predicate counting queries are a versatile class, consisting of queries that count the number of tuples satisfying any logical predicate.

Definition 1 (Predicate counting query). *A predicate on R is a boolean function $\phi : dom(R) \to \{0, 1\}$. A predicate can be used as a counting query on instance I of R whose answer is $\phi(I) = \sum_{t \in I} \phi(t)$.*

A predicate corresponds to a condition in the `WHERE` clause of an SQL statement, so in SQL a predicate counting query has the form: `SELECT Count(*) FROM R WHERE` ϕ.

When a predicate ϕ refers *only* to a subset of attributes $\mathcal{A} \subset attr(R)$ we may annotate the predicate, writing $[\phi]_{\mathcal{A}}$. If $[\phi_1]_{\mathcal{A}}$ and $[\phi_2]_{\mathcal{B}}$ are predicates on attribute sets $\mathcal{A}$ and $\mathcal{B}$, then their conjunction is a predicate $[\phi_1 \wedge \phi_2]_{\mathcal{A} \cup \mathcal{B}}$.

We assume that each query consists of *arbitrarily complex* predicates on each attribute, but require that they are combined across attributes with conjunctions. In other words, each ϕ is of the form $\phi = [\phi_1]_{A_1} \wedge \cdots \wedge [\phi_d]_{A_d}$. This facilitates the compact implicit representations described in Section 4. One approach to handling disjunctions (and other more complex query features) is to transform the schema by merging attributes. We illustrate this in its application to the SF1 workload, and return to this issue in Section 9.

Example 1. *The SF1 workload consists of conjunctive conditions over its attributes, with the exception of conditions on the six binary race attributes, which can be complex disjunctions of conjunctions (such as "The number of Persons with two or more races"). We simply merge the six binary race attributes and treat it like a single $2^6 = 64$ size attribute (called simply* Race*). This schema transformation does not change the overall domain size, but allows every SF1 query to be expressed as a conjunction.*

3.3 Logical view of query workloads

A workload is a set of predicate counting queries. A workload may consist of queries designed to support a variety of analyses or user needs, as is the case with the SF1 workload described above. Workloads may also be built from the sufficient statistics of models, or generated by tools that aid users in exploring data, or a combination of these analyses. For the privacy mechanisms considered here, it is preferable for the workload to explicitly mention all queries of interest, rather than a subset of the queries that could act like a supporting view, from which the remaining queries of interest could be computed. Enumerating all queries of interest allows error to be optimized collectively. In addition, a workload query can be repeated, or equivalently, weighted, to express the preference for greater accuracy on that query.

Structured multi-dimensional workloads. Multi-dimensional workloads are often defined in a structured form, as *products* and *unions of products*, that we will exploit later in our implicit representations. Following the notation above, we write $\Phi = [\phi_1 \dots \phi_p]_{\mathcal{A}}$ to denote a set of p predicates, each mentioning only attributes in $\mathcal{A}$. For example, the following are common predicate sets defined over a single attribute A of tuple t:

I	Identity_A	$= \{t.A == a_i \mid a_i \in dom(A)\}$
P	Prefix_A	$= \{(a_1 \le t.A \le a_i) \mid a_i \in dom(A)\}$
R	AllRange_A	$= \{(a_i \le t.A \le a_j) \mid a_i, a_j \in dom(A)\}$
T	Total_A	$= \{True\}$

Identity$_A$ contains one predicate for each element of the domain. Both Prefix$_A$ and Range$_A$ rely on an ordered $dom(A)$; they contain predicates defining a CDF (i.e. sufficient to compute the empirical cumulative distribution function), and the set of all range queries, respectively. The predicate set Total$_A$, consists of a single predicate, returning *True* for any $a \in dom(A)$, and thus counting all records.

We can construct multi-attribute workloads by taking the cross-product of predicate sets defined for single attributes, and *conjunctively* combining individual queries.

DEFINITION 2 (PRODUCT). *For predicate sets $\Phi = [\phi_1 \dots \phi_p]_{\mathcal{A}}$ and $\Psi = [\psi_1 \dots \psi_r]_{\mathcal{B}}$ ($\mathcal{A}$ and $\mathcal{B}$ are disjoint), the product is a query set containing a total of $p \cdot r$ queries:*

$$[\Phi \times \Psi]_{\mathcal{A}\cup\mathcal{B}} = \{\phi_i \wedge \psi_j | \phi_i \in \Phi, \psi_j \in \Psi\}$$

We describe several examples of workloads constructed from products and unions of products below.

EXAMPLE 2 (SINGLE QUERY AS PRODUCT). *A predicate counting query in the SF1 workload is:* `SELECT Count(*) FROM` Person `WHERE sex=M AND age < 5`. *We can express this query as a product: first, define predicate set $\Phi_1 = \{sex{=}M\}$ and predicate set $\Phi_2 = \{age < 5\}$. The query is expressed as the product $\Phi_1 \times \Phi_2$. (We omit Total on the other attributes for brevity.)*

EXAMPLE 3 (`GROUP BY` QUERY AS PRODUCT). *A* `GROUP BY` *query can be expressed as a product by including an Identity predicate set for each grouping attribute and a singleton predicate set for each attribute in the* `WHERE` *clause. The product would also include Total for each attribute not mentioned in the query. For example, the query* `SELECT sex, age, Count(*) FROM` Person `WHERE hispanic = TRUE GROUP BY sex, age` *is expressed as $I_{Sex} \times I_{Age} \times \Phi_3$ where $\Phi_3 = \{hispanic{=}True\}$. This product contains 2×115 counting queries, one for each possible setting of Sex and Age.*

EXAMPLE 4 (SF1 TABULATION AS PRODUCT). *Except for the population total, the queries in the P12 tabulation of the Census SF1 workload [6] can be described by a single product: $I_{\texttt{Sex}} \times R_{\texttt{Age}}$ where $R_{\texttt{Age}}$ is a particular set of range queries including $[0, 114], [0, 4], [5, 9], [10, 14], \dots [85, 114]$.*

Unions of products. Our workloads often combine multiple products as a union of the sets of queries in each product. For example, the set of all three-way marginals is a union of $\binom{d}{3}$ workloads, each a product of the Identity predicate set applied to three attributes.

The input to the algorithms that follow is a logical workload consisting of a union of products, each representing one or possibly many queries.

DEFINITION 3 (LOGICAL WORKLOAD). *A logical workload $\mathcal{W} = \{q_1 \dots q_k\}$ consists of a set of products q_i where each $q_i = [\Phi_{i1}]_{A_1} \times \dots \times [\Phi_{id}]_{A_d}$.*

EXAMPLE 5 (SF1 AS UNION OF PRODUCTS). *The SF1 workload from Section 2 can be represented in a logical form, denoted $\mathcal{W}_{SF1}$, that consists of a union of $k = 4151$ products, each representing a single query. Because these queries are at the national level, there is a Total predicate set on the State attribute. The logical form of the SF1+ workload, denoted $\mathcal{W}_{SF1+}$, includes those products, plus an additional 4151 products that are identical except for replacing the Total on State with an Identity predicate set. There are a total of $k = 8302$ products, representing a total of $4151{+}51{\times}4151 = 215{,}852$ predicate counting queries. While this is a direct translation from the SQL form, this representation can be reduced. First, we can reduce to $k = 4151$ products by simply adding True to the Identity predicate set on State to capture the national counts. Furthermore, through manual inspection, we found that both $\mathcal{W}_{SF1}$ and $\mathcal{W}_{SF1+}$ can be even more compactly represented as the union of 32 products—we use $\mathcal{W}^*_{SF1}$ and $\mathcal{W}^*_{SF1+}$ to denote more compact logical forms. This results in significant space savings (Example 7) and runtime improvements.*

3.4 Explicit data and query vectorization

The vector representation of predicate counting queries (and the data they are evaluated on) is central to the select-measure-reconstruct paradigm. The vector representation of instance I is denoted $\boldsymbol{x}_I$ (or simply $\boldsymbol{x}$ if the context is clear) and called the *data vector*.[2] Each entry in $\boldsymbol{x}_I$ corresponds to a tuple $t \in dom(R)$ and reports the number of occurrences of t in I. Note that, throughout the paper, the representation of the data vector is *always* explicit; it is the representation of queries that will be implicit.

Every predicate counting query ϕ has a vector form.

DEFINITION 4 (VECTORIZED QUERY). *Given a predicate counting query ϕ defined on schema R, its vectorization is denoted $vec(\phi)$ and has an entry equal to $\phi(t) \in \{0, 1\}$ for each tuple $t \in dom(R)$.*

The above definition immediately suggests a simple algorithm for computing $vec(\phi)$: form a vector by evaluating ϕ on each element of the domain and recording the 0 or 1 output of evaluation. (A more efficient algorithm is presented in the next section.) Note that both the data vector and the vectorized query have size $|dom(R)| = N$. Once a predicate query is vectorized, it can easily be evaluated by taking its dot product with the data vector: that is, $\phi(I) = vec(\phi) \cdot \boldsymbol{x}_I$ for any instance I.

A single predicate counting query is represented as a vector, so a workload of predicate counting queries can be represented as a matrix in which queries are rows. For logical workload $\mathcal{W}$, its (explicit) matrix form is written $\boldsymbol{W}$, and the evaluation of the workload is equivalent to the matrix product $\boldsymbol{W}\boldsymbol{x}_I$. Note that the size of the workload matrix is $m \times N$ where m is the number of queries, $\boldsymbol{x}_I$ is $N \times 1$, and the vector of workload answers is $m \times 1$.

3.5 Differential privacy

Differential privacy is a property of a randomized algorithm that bounds the difference in output probabilities induced by changes to an individual's data. Let $nbrs(I)$ be the set of databases differing from I in at most one record.

DEFINITION 5 (DIFFERENTIAL PRIVACY [11]). *A randomized algorithm $\mathcal{K}$ is (ϵ, δ)-differentially private if for any instance I, any $I' \in nbrs(I)$, and any outputs $O \subseteq Range(\mathcal{K})$,*

$$Pr[\mathcal{K}(I) \in O] \leq \exp(\epsilon) \times Pr[\mathcal{K}(I') \in O] + \delta$$

[2]When R has d attributes, the data vector has a multi-dimensional interpretation as a d-way array, or a tensor; to simplify notation we assume appropriate flattening.

We focus exclusively on ϵ-differential privacy (i.e. $\delta = 0$). However our techniques also apply to a version of MM satisfying approximate differential privacy ($\delta > 0$) [28].

The Laplace mechanism underlies the private mechanisms considered in this paper; we describe it in vector form. Let $\text{Lap}(\sigma)^m$ denote a vector of m independent samples from a Laplace distribution with mean 0 and scale σ.

DEFINITION 6 (LAPLACE MECHANISM, VECTOR FORM). *Given an $m \times N$ query matrix $\boldsymbol{A}$, the randomized algorithm LM that outputs the following vector is ϵ-differentially private: $LM(\boldsymbol{A}, \boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + Lap(\sigma_{\boldsymbol{A}})^m$ where $\sigma_{\boldsymbol{A}} = \frac{\|\boldsymbol{A}\|_1}{\epsilon}$.*

Above, $\|\boldsymbol{A}\|_1$ denotes the maximum absolute column sum norm of $\boldsymbol{A}$, shown in [25] to be equal to the *sensitivity* of the query set defined by $\boldsymbol{A}$, since it measures the maximum difference in the answers to the queries in $\boldsymbol{A}$ on any two databases that differ only by a single record.

For an algorithm $\mathcal{K}$ answering workload $\mathcal{W}$, we measure error as the expected total squared error on the workload query answers, denoted $Err(\mathcal{W}, \mathcal{K})$.

3.6 The matrix mechanism

For a workload $\boldsymbol{W}$ in matrix form, defined on data vector $\boldsymbol{x}$, the matrix mechanism [25] is defined in Table 1(a). Privacy of the matrix mechanism (and thus all techniques in this paper) follows from the privacy of the Laplace Mechanism (output $\boldsymbol{y}$ of the MEASURE step). The RECONSTRUCT steps (inference and workload answering) perform post-processing on $\boldsymbol{y}$, so they do not degrade privacy [12].

We use expected total squared error as the error metric and optimization objective. This is the same error metric proposed originally by the matrix mechanism, as well as a number of other works [19, 24, 28, 34, 41].

DEFINITION 7 (WORKLOAD ERROR UNDER STRATEGY). *Given workload matrix $\boldsymbol{W}$ and strategy $\boldsymbol{A}$, the expected total squared error of the workload query answers is:*

$$Err(\boldsymbol{W}, MM(\boldsymbol{A})) = \frac{2}{\epsilon^2} \|\boldsymbol{A}\|_1^2 \, ||\boldsymbol{W}\boldsymbol{A}^+||_F^2$$

This error metric has a number of advantages: it is independent of the input data and the setting of ϵ, and it can be computed in closed form [25]. As a result, if the workload is fixed,[3] the optimized strategy $\boldsymbol{A}$ can be computed once and used for multiple invocations of measure and reconstruct (i.e. on different input datasets and/or for different outputs generated with different ϵ values).

This error metric is an *absolute* measure of error, as opposed to a *relative* measure of error, which would report error normalized by the actual query answer. The techniques in this paper are not applicable to relative error measures; the objective function of the strategy selection problem would depend on the input data, and we would need to solve for the best strategy for a workload *and* dataset.

4. IMPLICIT REPRESENTATIONS

Workload matrices can be represented implicitly, in a form that is far more concise than materialized explicit workload matrices, while still allowing key operations to be performed.

[3]E.g., the Census SF1 workload is determined for each decennial census and therefore changes only every 10 years.

4.1 Implicitly vectorized conjunctions

Consider a predicate defined on a single attribute, A_1, where $|dom(A_1)| = n_1$. This predicate, $[\phi]_{A_1}$, can be vectorized with respect to just the domain of A_1 (and not the full domain of all attributes) similarly to Definition 4. When a predicate is formed from the conjunction of such single-attribute predicates, its vectorized form has a concise implicit representation in terms of the *kronecker product*, denoted $\otimes$, between vectors. (Here we treat a length n vector as an $1 \times n$ matrix.)

DEFINITION 8 (KRONECKER PRODUCT). *For two matrices $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{m' \times n'}$, their Kronecker product $\boldsymbol{A} \otimes \boldsymbol{B} \in \mathbb{R}^{m \cdot m' \times n \cdot n'}$ is:*

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{11}\boldsymbol{B} & \cdots & a_{1n}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\boldsymbol{B} & \cdots & a_{mn}\boldsymbol{B} \end{bmatrix}$$

THEOREM 1 (IMPLICIT VECTORIZATION). *Let $\phi = [\phi_1]_{A_1} \wedge [\phi_2]_{A_2}$ be a predicate defined by the conjunction of ϕ_1 and ϕ_2 on attributes A_1 and A_2. Then, $vec(\phi) = vec(\phi_1) \otimes vec(\phi_2)$.*

While the explicit representation of $vec(\phi)$ has size $n_1 \cdot n_2$, the implicit representation, $vec(\phi_1) \otimes vec(\phi_2)$, requires storing only $vec(\phi_1)$ and $vec(\phi_2)$, which has size $n_1 + n_2$. More generally, for a predicate counting query that is a conjunction of predicates on each of the d attributes, the implicit vectorized representation of the query has size just $\Sigma_{i=1}^{d} n_i$ while the explicit representation has size $\Pi_{i=1}^{d} n_i$.

EXAMPLE 6. *Recall that the workload $\mathcal{W}_{SF1}$ consists of 4151 queries, each defined on a data vector of size* 500,480*. Since explicitly vectorized queries are the same size as the domain, the size of the explicit workload matrix is* $4151 \times$ 500,480*, or* 8.3*GB. Using the implicit representation, each query can be encoded using* $2+2+64+115+17 = 200$ *values, for a total of 3.3MB. For $\mathcal{W}_{SF1+}$, which consists of* 215,852 *queries on a data vector of size* 25,524,480*, the explicit workload matrix would require 22TB of storage. In contrast, the implicit vector representation would require 200MB.*

4.2 Implicitly vectorized products

The kronecker product can naturally encode product workloads too (as in Definition 2). Given a (logical) product $\Phi \times \Psi$, we can implicitly represent its queries as a Kronecker product of two matrices: one representing the predicates in Φ and one representing the predicates in Ψ. If Φ has p predicates, and the vector form of each predicate has size $n_1 = |dom(A_1)|$, it is represented (explicitly) as a $p \times n_1$ matrix. If Ψ contains r predicates of size $n_2 = |dom(A_2)|$, it is similarly represented as an $r \times n_2$ matrix. We can store only these matrices, implicitly representing the product as the Kronecker product:

THEOREM 2. *Given predicate sets $\Phi = [\phi_1 \dots \phi_p]_{A_1}$ and $\Psi = [\psi_1 \dots \psi_r]_{A_2}$, on attributes A_1 and A_2, the vectorized product is defined in terms of matrices $vec(\Phi)$ and $vec(\Psi)$: $vec(\Phi \times \Psi) = vec(\Phi) \otimes vec(\Psi)$.*

The size of the implicit representation is $pn_1 + rn_2$, while the explicit product has size prn_1n_2.

4.3 Workload encoding algorithm

Given as input a logical workload $\mathcal{W}$ (as in Definition 3), the ImpVec algorithm produces an implicitly represented workload matrix with the following form:

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

Here stacking sub-workloads is analogous to union and in formulas we will write an implicit union-of-products workload as $\mathbb{W}_{[k]} = w_1\mathbb{W}_1 + \ldots + w_k\mathbb{W}_k$. We use blackboard bold font to distinguish an implicitly represented workload $\mathbb{W}$ from an explicitly represented workload $\boldsymbol{W}$. Note that line

Algorithm 1: ImpVec

Input: Workload $\mathcal{W} = \{q_1 \dots q_k\}$ and weights $w_1 \dots w_k$
Output: Implicit workload $\mathbb{W}_{[k]}$

1. For each product $q_i \in \mathcal{W}$: $q_i = [\Phi_{i1}]_{A_1} \times \cdots \times [\Phi_{id}]_{A_d}$
2. For each $j \in [1..d]$
3. compute $\boldsymbol{W}_j^{(i)} = vec(\Phi_{ij})$
4. Let $\mathbb{W}_i = \boldsymbol{W}_1^{(i)} \otimes \cdots \otimes \boldsymbol{W}_d^{(i)}$
5. **Return:** $w_1\mathbb{W}_1 + \ldots + w_k\mathbb{W}_k$

3 of the ImpVec algorithm is *explicit* vectorization, as in Definition 4, of a set of predicates on a single attribute.

Example 7. *Recall from Example 5 that the* 215,852 *queries of* $\mathcal{W}_{SF1+}$ *can be represented as* $k = 8032$ *products. If* $\mathcal{W}_{SF1+}$ *is represented in this factored form, the* ImpVec *algorithm returns a smaller implicit representation, reducing the 200MB (from Example 6) to 50 MB. If the workloads are presented in their manually factored format of* $k = 32$ *products, the implicit representation of* $\mathcal{W}^*_{SF1}$ *requires only 335KB, and* $\mathcal{W}^*_{SF1+}$ *only 687KB.*

4.4 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 [37] accelerate strategy selection and reconstruction. For strategy selection, a critical object is the matrix product $\mathbb{W}^T\mathbb{W}$. When $\mathbb{W} = \boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d$, then $\mathbb{W}^T\mathbb{W} = \boldsymbol{W}_1^T\boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d^T\boldsymbol{W}_d$. For the inference phase of reconstruction, computing the pseudo-inverse $\mathbb{A}^+$ is the main challenge. If $\mathbb{A} = \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$, then $\mathbb{A}^+ = \boldsymbol{A}_1^+ \otimes \cdots \otimes \boldsymbol{A}_d^+$. Lastly, implicit strategy matrices allow for straightforward calculation of sensitivity:

Theorem 3 (Sensitivity of Kronecker product). *Given an implicitly defined strategy matrix* $\boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$, *its sensitivity is:* $||\boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d||_1 = \prod_{i=1}^d ||\boldsymbol{A}_i||_1$

Note that sparse matrices can also provide very compact representations but do not support key operations in implicit form; for example, $\boldsymbol{W}^T\boldsymbol{W}$ may not be sparse even if $\boldsymbol{W}$ is.

5. OPTIMIZING EXPLICIT WORKLOADS

In this section we develop a solution to the strategy selection problem that works for explicitly-represented workloads and scales to modest domain sizes (of about 10^4). This method, denoted OPT_0, is a sub-routine used in Section 6.

Below we state the optimization problem underlying strategy selection and introduce gradient-based optimization. We then describe a careful restriction of the strategy space required to make gradient-based optimization scalable and effective, leading to the definition of OPT_0.

5.1 The optimization problem

Our goal is to find a strategy $\boldsymbol{A}$ that (a) *supports* the input workload $\boldsymbol{W}$ while (b) offering minimum expected total squared error as per Definition 7. Strategy $\boldsymbol{A}$ supports a workload $\boldsymbol{W}$ if and only if every query in $\boldsymbol{W}$ can be expressed as a linear combination of the queries in $\boldsymbol{A}$, which occurs whenever $\boldsymbol{W} = \boldsymbol{W}\boldsymbol{A}^+\boldsymbol{A}$ [28]. The expected total squared error of the workload using a strategy with sensitivity 1[4] is equal to $||\boldsymbol{W}\boldsymbol{A}^+||_F^2$, where $||\cdot||_F$ is the Frobenius norm. The resulting constrained optimization problem is:

Problem 1. *Given an* $m \times n$ *workload matrix* $\boldsymbol{W}$*:*

$$\begin{aligned} &\underset{\boldsymbol{A} \in \mathbb{R}^{p \times n}}{\text{minimize}} && ||\boldsymbol{W}\boldsymbol{A}^+||_F^2 \\ &\text{subject to} && \boldsymbol{W}\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{W} \ , \ ||\boldsymbol{A}||_1 \leq 1 \end{aligned} \quad (2)$$

This optimization problem is difficult to solve exactly. It has many variables and is not convex, both the objective function and constraints involve $\boldsymbol{A}^+$, which can be slow to compute, and, in addition, the constraint on $||\boldsymbol{A}||_1$ is not differentiable. Finally, $||\boldsymbol{W}\boldsymbol{A}^+||_F^2$ has points of discontinuity near the boundary of the constraint $\boldsymbol{W}\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{W}$. This problem was originally formulated as a rank-constrained semi-definite program [25], which will converge to the global optimum, but requires $O(m^4(m^4 + N^4))$ time, making it infeasible for practical cases.

Gradient-based numerical optimization techniques can be used to find locally optimal solutions. These techniques begin by guessing a solution $\boldsymbol{A}_0$ and then iteratively improving 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. A direct application of gradient-based optimization methods does not work due to the constraints, and so a projected gradient method must be used instead. Even without the constraints, gradient-based optimization is slow, as the cost of computing the objective function for general $\boldsymbol{A}$ is $O(N^3)$, e.g. requiring more than 6 minutes for $N = 8192$.

5.2 Parameterized optimization: OPT_0

We now present an algorithm to solve Problem 1 by judiciously restricting the search space of the optimization problem. Recall that our goal is to search over $\boldsymbol{A}$ that support the workload $\boldsymbol{W}$. The key observation we make is the following: if $\boldsymbol{A}$ contains one query that counts the number of records in the database for each domain element in $dom(R)$ – i.e. $\boldsymbol{A}$ contains a scaled identity matrix – then any workload $\boldsymbol{W}$ is supported by $\boldsymbol{A}$. Of course $\boldsymbol{A} = \boldsymbol{I}$ is not a good strategy for many workloads (e.g., this has poor error for a workload encoding prefix queries). Hence, we search over strategies that contain a scaled identity matrix in addition to p extra rows, as defined below.

Definition 9 (p-Identity strategies). *Given a* $p \times N$ *matrix of non-negative values* $\boldsymbol{\Theta}$*, the p-Identity strategy matrix* $\boldsymbol{A}(\boldsymbol{\Theta})$ *is defined as follows:*

[4]Li et al [25] showed that error-optimal strategy matrices have equal L_1 column norm, which can be normalized to 1.

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

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

Above, $\boldsymbol{D}$ is a diagonal matrix that scales the columns of $\boldsymbol{A}(\boldsymbol{\Theta})$ so that $||\boldsymbol{A}||_1 = 1$. The $p \times N$ values in $\boldsymbol{\Theta}$ determine the weights of the p queries as well as the weights on the identity queries (where a lower weight query will be answered with greater noise).

EXAMPLE 8. *For $p = 2$ and $N = 3$, we illustrate below how $\boldsymbol{A}(\boldsymbol{\Theta})$ is related to its parameter matrix, $\boldsymbol{\Theta}$.*

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

For this class of parameterized strategies, the resulting optimization problem is stated below; we use OPT_0 to denote the operator that solves this problem.

PROBLEM 2 (PARAMETERIZED OPTIMIZATION). *Given workload matrix $\boldsymbol{W}$ and hyper-parameter p:*

$$\underset{\{\boldsymbol{A}=\boldsymbol{A}(\boldsymbol{\Theta}) \mid \boldsymbol{\Theta} \in \mathbb{R}_+^{p \times n}\}}{minimize} \quad ||\boldsymbol{W}\boldsymbol{A}^+||_F^2$$

This parameterization was carefully designed to provide the following beneficial properties:

Constraint resolution Problem 2 is a simpler optimization problem than Problem 1 because it is unconstrained: $\boldsymbol{W}\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{W}$ and $||\boldsymbol{A}||_1 \leq 1$ are satisfied for all $\boldsymbol{A} = \boldsymbol{A}(\boldsymbol{\Theta})$.

Expressiveness Considering only p-Identity strategy matrices could mean that we omit from consideration the optimal strategy, but it also reduces the number of variables, allowing more effective gradient-based optimization. The expressiveness depends on the parameter p used to define matrices $\boldsymbol{A}(\boldsymbol{\Theta})$, which is an important hyper-parameter. In practice we have found $p \approx \frac{n}{16}$ to provide a good balance between efficiency and expressiveness for complex workloads (such as the set of all range queries). For less complex workloads, an even smaller p may be used.

Efficient gradient, objective, and inference To a first approximation, the runtime of gradient-based optimization is #RESTARTS∗#ITER∗(COST$_{grad}$+COST$_{obj}$), where COST$_{grad}$ is the cost of computing the gradient, COST$_{obj}$ is the cost of computing the objective function, #RESTARTS is the number of random restarts, and #ITER is the number of iterations per restart. For strategy $\boldsymbol{A}$, the objection function, denoted $C(\boldsymbol{A})$, and the gradient function, denoted $\frac{\partial C}{\partial \boldsymbol{A}}$, are defined:

$$C(\boldsymbol{A}) = ||\boldsymbol{W}\boldsymbol{A}^+||_F^2 = tr[(\boldsymbol{A}^T\boldsymbol{A})^+(\boldsymbol{W}^T\boldsymbol{W})] \tag{3}$$

$$\frac{\partial C}{\partial \boldsymbol{A}} = -2\boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^+(\boldsymbol{W}^T\boldsymbol{W})(\boldsymbol{A}^T\boldsymbol{A})^+ \tag{4}$$

Note that the above expressions depend on $\boldsymbol{W}$ through the matrix product $\boldsymbol{W}^T\boldsymbol{W}$, which can be computed once and cached for all iterations and restarts; we therefore omit this cost from complexity expressions. It always has size $N \times N$ regardless of the number of queries in $\boldsymbol{W}$. For highly structured workloads (e.g all range queries), $\boldsymbol{W}^T\boldsymbol{W}$ can be computed directly without materializing $\boldsymbol{W}$, so we allow OPT_0 to take $\boldsymbol{W}^T\boldsymbol{W}$ as input in these special cases.

For general $\boldsymbol{A}$, the runtime complexity of computing $C(\boldsymbol{A})$ and $\frac{\partial C}{\partial \boldsymbol{A}}$ is $O(N^3)$. By exploiting the special structure of $\boldsymbol{A}(\boldsymbol{\Theta})$, we can reduce these costs to $O(pN^2)$:

THEOREM 4 (COMPLEXITY OF OPT_0). *Given any p-Identity strategy $\boldsymbol{A}(\Theta)$, both the objective function $C(\boldsymbol{A}(\boldsymbol{\Theta}))$ and the gradient $\frac{\partial C}{\partial \boldsymbol{A}}$ can be evaluated in $O(pN^2)$ time.*

The speedup resulting from this parameterization is often much greater in practice than the $\frac{N}{p}$ improvement implied by the theorem. When $N = 8192$, computing the objective for general $\boldsymbol{A}$ takes > 6 minutes, while it takes only 1.5 seconds for a p-Identity strategy: a 240× improvement. Nevertheless, OPT_0 is only practical for modest domain sizes ($N \sim 10^4$). For multi-dimensional data we do not use it directly, but as a subroutine in the algorithms to follow.

6. OPTIMIZING IMPLICIT WORKLOADS

We present next a series of optimization techniques for multi-dimensional workloads, exploiting the implicit workload representations presented in Section 4. One of the main ideas is to decompose a strategy optimization problem on a multi-dimensional workload into a sequence of optimization problems on individual attributes. The optimization operators $\mathsf{OPT}_\otimes$, OPT_+, and OPT_M are each variants described below and are summarized in Table 2.

6.1 Optimizing product workloads

Recall that $\mathbb{W} = \boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d$, for an implicit workload that is the product of d sub-workloads. If $\boldsymbol{W}_i$ is defined with respect to a domain of size n_i, then $\mathbb{W}$ is defined on a total domain of size of $N = \prod_{i=1}^d n_i$. We perform strategy optimization directly on this implicit product representation by decomposing the optimization problem into a series of explicit optimizations on the sub-workloads.

DEFINITION 10 ($\mathsf{OPT}_\otimes$). *Given workload $\mathbb{W}$ and parameter vector $\vec{p} = \langle p_1 \dots p_d \rangle$, the operator $\mathsf{OPT}_\otimes(\mathbb{W}, \vec{p})$ applies OPT_0 to each sub-workload and returns a product strategy:*

$$\begin{aligned} \mathsf{OPT}_\otimes(\mathbb{W}, \vec{p}) &= \mathsf{OPT}_\otimes(\boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d, \vec{p}) \\ &\overset{def}{=} \mathsf{OPT}_0(\boldsymbol{W}_1, p_1) \otimes \cdots \otimes \mathsf{OPT}_0(\boldsymbol{W}_d, p_k) \\ &= \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d = \mathbb{A} \end{aligned}$$

Therefore $\mathsf{OPT}_\otimes$ merely requires solving d independent instances of OPT_0 and the resulting output strategy is the product of d distinct p_i-Identity strategies. This decomposition has a well-founded theoretical justification. Namely, if we restrict the solution space to a (single) product strategy, so that $\boldsymbol{A}$ has the form $\mathbb{A} = \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$, then the error of the workload under $\mathbb{A}$ decomposes into the product of the errors of its sub-workloads under corresponding sub-strategies. Thus overall error is minimized when $Err(\boldsymbol{W}_i, \boldsymbol{A}_i)$ is minimized for each i and it follows that the $\mathsf{OPT}_\otimes$ program minimizes the correct objective function.

THEOREM 5 (ERROR DECOMPOSITION). *Given a workload $\mathbb{W} = \boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d$ and a sensitivity 1 strategy $\mathbb{A} = \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$, the error is proportional to:*

$$||\mathbb{W}\mathbb{A}^+||_F^2 = \prod_{i=1}^d ||\boldsymbol{W}_i\boldsymbol{A}_i^+||_F^2$$

The cost of each iteration in $\mathsf{OPT}_\otimes(\mathbb{W})$ is the sum of costs of d independent optimizations of the sub-workloads $\boldsymbol{W}_i$.

Table 2: Summary of optimization operators: input and output types, and the complexity of objective/gradient functions.

Definition		Output strategy	Optimization Operator			Input workload	Complexity
§5.2	Problem 2	p-Identity matrix	$\boldsymbol{A}(\boldsymbol{\Theta})$	$\leftarrow$	$\mathsf{OPT}_0(\boldsymbol{W}, p)$	Any explicit $\boldsymbol{W}$	$N^2 p$
§6.1	Definition 10	Product, $A(\boldsymbol{\Theta})$ terms	$\mathbb{A}$	$\leftarrow$	$\mathsf{OPT}_\otimes(\mathbb{W}, \vec{p})$	Single product	$\sum_{i=1}^d n_i^2 p_i$
§6.2	Problem 3	Product, $A(\boldsymbol{\Theta})$ terms	$\mathbb{A}$	$\leftarrow$	$\mathsf{OPT}_\otimes(w_1\mathbb{W}_1 + \cdots + w_k\mathbb{W}_k, \vec{p})$	Union of products	$k\sum_{i=1}^d n_i^2 p_i$
§6.2	Definition 11	Union of products	$\mathbb{A}_1 + \cdots + \mathbb{A}_l$	$\leftarrow$	$\mathsf{OPT}_+(\mathbb{W}_{[k_1]}, \ldots, \mathbb{W}_{[k_l]}, \vec{p})$	Union of products	$k\sum_{i=1}^d n_i^2 p_i$
§6.3	Problem 4	Weighted marginals	$\mathbb{M}(\boldsymbol{\theta})$	$\leftarrow$	$\mathsf{OPT}_\mathsf{M}(w_1\mathbb{W}_1 + \cdots + w_k\mathbb{W}_k)$	Union of products	4^d

Since the cost of each iteration in OPT_0 is $O(p_i n_i^2)$, the cost for $\mathsf{OPT}_\otimes(\mathbb{W})$ is $O(\sum p_i n_i^2)$. Note that if $\mathbb{W}$ were represented explicitly, each iteration in OPT_0 would take $O(\prod_i p_i n_i^2)$.

6.2 Optimizing unions of product workloads

We now define three approaches for optimizing implicit workloads that are (weighted) unions of products. Each approach restricts the strategy to a different region of the full strategy space for which optimization is tractable (see Table 2). The first computes a strategy consisting of a single product; it generalizes $\mathsf{OPT}_\otimes$. The second, OPT_+, can generate strategies consisting of unions of products. The third, OPT_M, generates a strategy of weighted marginals.

Single-product output strategy For weighted union of product workloads, if we restrict the optimization problem to a single product strategy, then the objective function decomposes as follows:

THEOREM 6. *Given workload $\mathbb{W}_{[k]} = w_1\mathbb{W}_1 + \ldots + w_k\mathbb{W}_k$ and strategy matrix $\mathbb{A} = \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$, workload error is:*

$$||\mathbb{W}_{[k]}\mathbb{A}^+||_F^2 = \sum_{j=1}^k w_j^2 \prod_{i=1}^d ||\boldsymbol{W}_i^{(j)}\boldsymbol{A}_i^+||_F^2 \tag{5}$$

This leads to the following optimization problem:

PROBLEM 3 (UNION OF PRODUCT OPTIMIZATION). *For a workload $\mathbb{W}_{[k]} = w_1\mathbb{W}_1 + \ldots + w_k\mathbb{W}_k$ and parameter vector $\vec{p} = \langle p_1 \ldots p_d \rangle$, let:*

$$\boldsymbol{A}_i = \underset{\{\boldsymbol{A}_i(\boldsymbol{\Theta}_i) \mid \boldsymbol{\Theta}_i \in \mathbb{R}_+^{p_i \times n}\}}{\text{minimize}} \quad \sum_{j=1}^k w_j^2 \prod_{i'=1}^d ||\boldsymbol{W}_{i'}^{(j)}\boldsymbol{A}_{i'}^+||_F^2$$

and form final solution strategy as $\mathbb{A} = \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$.

When $k = 1$, Problem 3 returns the same solution as Definition 10, so we use matching notation and allow the $\mathsf{OPT}_\otimes$ operator to accept a single product or a union of products, as shown in Table 2. Problem 3 is a coupled optimization problem, and we use a block method that cyclically optimizes $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_d$ until convergence. We begin with random initializations for all $\boldsymbol{A}_i$. We optimize one $\boldsymbol{A}_i$ at a time, fixing the other $\boldsymbol{A}_{i'} \neq \boldsymbol{A}_i$ using OPT_0 on a carefully constructed surrogate workload $\hat{\boldsymbol{W}}_i$ (equation 6) that has the property that the error of any strategy $\boldsymbol{A}_i$ on $\hat{\boldsymbol{W}}_i$ is the same as the error of $\mathbb{A}$ on $\mathbb{W}$. Hence, the correct objective function is optimized.

$$\hat{\boldsymbol{W}}_i = \begin{bmatrix} c_1\boldsymbol{W}_i^{(1)} \\ \vdots \\ c_k\boldsymbol{W}_i^{(k)} \end{bmatrix} \quad c_j = w_j \prod_{i' \neq i} ||\boldsymbol{W}_{i'}^{(j)}\boldsymbol{A}_{i'}^+||_F \tag{6}$$

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 sub-problems allows OPT_0 to be efficient.

Union-of-products output strategy For certain workloads, restricting to solutions consisting of a single product, as $\mathsf{OPT}_\otimes$ does, excludes good strategies. This can happen for a workload like $W = (R \times T) \cup (T \times R)$, for which choosing a single product tends to force a suboptimal pairing of queries across attributes. Unfortunately, we cannot optimize directly over union-of-product strategies because computing the expected error is intractable. Nevertheless, we can use our existing optimization methods to generate high-quality union-of-product strategies. This operator takes as input a weighted union of products, partitioned into l subsets. It optimizes each individually using $\mathsf{OPT}_\otimes$ and combines the resulting output strategies to form a strategy consisting of a union of l products. Below we use $K = k_1 + \cdots + k_l$ (and recall notation $\mathbb{W}_{[k]}$ from Section 4.3):

DEFINITION 11 (OPT_+). *Given a workload, $\mathbb{W}_{[K]} = \mathbb{W}_{[k_1]} + \cdots + \mathbb{W}_{[k_l]}$, and parameter vector $\vec{p}$, the optimization routine OPT_+ returns the union of strategies defined below:*

$$\begin{aligned}\mathsf{OPT}_+(\mathbb{W}_{[K]}, \vec{p}) &\stackrel{def}{=} \mathsf{OPT}_\otimes(\mathbb{W}_{[k_1]}, \vec{p}) + \cdots + \mathsf{OPT}_\otimes(\mathbb{W}_{[k_l]}, \vec{p}) \\ &= \mathbb{A}_1 + \cdots + \mathbb{A}_l\end{aligned}$$

This definition could easily be extended so that each $\mathbb{A}_i$ gets a different fraction of the privacy budget, and so that each call to $\mathsf{OPT}_\otimes$ gets a different parameter vector.

6.3 Optimized marginal strategies

Although $\mathsf{OPT}_\otimes$ or OPT_+ can be used to optimize workloads consisting of marginals, we now describe a third optimization operator, OPT_M, which is especially effective for marginal workloads (but is applicable to any union of product workload). A single marginal can be specified by a subset of attributes $S \subseteq [d]$ and can be expressed as the product $\boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$ where $\boldsymbol{A}_i = \boldsymbol{I}$ if $i \in S$ and $\boldsymbol{A}_i = \boldsymbol{T}$ otherwise. Since there are 2^d subsets of $[d]$, a set of weighted marginals can be characterized by a vector $\boldsymbol{\theta}$ of 2^d non-negative weights where θ_1 is the weight on the 0-way marginal (i.e., the total query) and θ_{2^d} is the weight on the d-way marginal (i.e., the queries defining the full contingency table). We use $\mathbb{M}(\boldsymbol{\theta})$ to denote the matrix which stacks each of these 2^d weighted marginals. Each marginal has sensitivity 1 so the total sensitivity of $\mathbb{M}(\boldsymbol{\theta})$ is $\sum \theta_i$. We resolve the sensitivity constraint from Problem 1 by moving it into the objective function, and we resolve the other constraint by forcing θ_{2^d} to be strictly positive. The resulting optimization problem is given below:

PROBLEM 4 (MARGINALS OPTIMIZATION). *Given a workload, $\mathbb{W}_{[k]} = w_1\mathbb{W}_1 + \ldots + w_k\mathbb{W}_k$, let:*

$$\boldsymbol{\theta} = \underset{\boldsymbol{\theta} \in \mathbb{R}_+^{2^d}; \theta_{2^d} > 0}{\text{minimize}} \quad \Big(\sum \theta_i\Big)^2 ||\mathbb{W}_{[k]}\mathbb{M}(\boldsymbol{\theta})^+||_F^2$$

Without materializing $\mathbb{M}(\boldsymbol{\theta})$ we can evaluate the objective function and its gradient by exploiting the structure of $\mathbb{M}(\boldsymbol{\theta})$. For strategies of the form $\mathbb{M}(\boldsymbol{\theta})$, $(\mathbb{M}^T\mathbb{M})^+$ can be written as the weighted sum of 2^d kronecker products, where each sub-matrix is either $\boldsymbol{I}$ or $\mathbf{1} = \boldsymbol{T}^T\boldsymbol{T}$, and we can efficiently find the 2^d weights that characterize the inverse by solving a (sparse) linear system of equations. The objective function only depends on $\mathbb{W}_{[k]}$ through the trace and sum of $(\boldsymbol{W}^T\boldsymbol{W})_i^{(j)}$. Thus, these statistics can be precomputed for the workload and the objective function can be evaluated very efficiently. The cost of this precomputation is linear in k, but subsequent to that, evaluating the objective and gradient only depends on d, and not n_i or k. Specifically, the time complexity of evaluating the objective and gradient is $O(4^d)$ – quadratic in 2^d, the size of $\boldsymbol{\theta}$.

7. THE HDMM ALGORITHM

The complete HDMM algorithm is described in Table 1(b). Here we explain $\mathsf{OPT}_{\mathsf{HDMM}}$, efficient MEASURE and RECONSTRUCT methods, and conclude with a privacy statement.

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

Using the optimization operators defined in previous sections, we now define $\mathsf{OPT}_{\mathsf{HDMM}}$, our fully-automated strategy selection algorithm. Predicting which optimization operator will yield the lowest error strategy requires domain expertise and may be challenging for complex workloads. Since our operators are usually efficient, we simply run multiple operators, keeping the output strategy that offers least error. We emphasize that strategy selection is independent of the input data and does not consume the privacy budget.

The algorithm below takes as input: implicit workload $\mathbb{W}$, operator set $\mathcal{P}$, and the maximum number of restarts, S.

Algorithm 2: $\mathsf{OPT}_{\mathsf{HDMM}}$
Input: implicit workload $\mathbb{W}$, operator set $\mathcal{P}$, max-restarts S
Output: implicit strategy $\mathbb{A}$

1. $best = (\boldsymbol{I}, error_{\boldsymbol{I}})$
2. **For** random starts $[1..S]$:
3. **For** each operator $P_i \in \mathcal{P}$:
4. $(\mathbb{A}_i, error_i) = P_i(\mathbb{W})$
5. **if** $error_i < best$ **emit** $\mathbb{A}_i$ and **update** *best*.

We instantiate $\mathsf{OPT}_{\mathsf{HDMM}}$ with an operator set consisting of: $\mathsf{OPT}_{\otimes}(\mathbb{W}, \vec{p})$, $\mathsf{OPT}_{+}(g(\mathbb{W}), \vec{p})$, and $\mathsf{OPT}_{\mathsf{M}}(\mathbb{W})$. ($\mathsf{OPT}_0$ does not appear here explicitly because it is called by $\mathsf{OPT}_{\otimes}$ and OPT_{+}). We use the following convention for setting the p parameters: if an attribute's predicate set is contained in $T \cup I$, 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. In OPT_{+} above, g forms two groups from the unioned terms in $\mathbb{W}$.

Extensions to the above algorithm—e.g., cost-based exploration of the operator space, or an extended operator set with alternative parameter settings—could improve upon the already significant improvements in accuracy we report in our experimental evaluation and are left for future work.

7.2 Efficient MEASURE and RECONSTRUCT

The implicit form of the output strategies enables efficiency for the **measure** and **reconstruct** stages. Recall from Section 6 that $\mathsf{OPT}_{\otimes}$ returns a product strategy and OPT_{+} returns a union of products, and in both cases, the d terms in each product are p-Identity matrices. Similarly, $\mathsf{OPT}_{\mathsf{M}}$ returns a union of products consisting of the marginals building blocks $\boldsymbol{T}$ and $\boldsymbol{I}$.

To exploit this structure to accelerate the MEASURE phase we define an efficient $\mathsf{Multiply}(\mathbb{A}, \boldsymbol{x})$ operation (as in Table 1(b)) for $\mathbb{A} = \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$ (although it is easily extended to a union of kronecker products.) The key property of Kronecker products that we need is given in Equation 7:

$$(\boldsymbol{B} \otimes \boldsymbol{C})\,\mathrm{flat}(\boldsymbol{X}) = \mathrm{flat}(\boldsymbol{B}\boldsymbol{X}\boldsymbol{C}^T) \tag{7}$$

where $\mathrm{flat}(\cdot)$ "flattens" a matrix into a vector by stacking the (transposed) rows into a column vector. The expression on the right is computationally tractable, as it avoids materializing the potentially large matrix $\boldsymbol{B} \otimes \boldsymbol{C}$. Matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ need not be explicitly represented if we can compute matrix-vector products with them. Thus, by setting $\boldsymbol{B} \leftarrow \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_{d-1}$ and $\boldsymbol{C} \leftarrow \boldsymbol{A}_d$ we can compute matrix-vector products for matrices of the form $\boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$ efficiently. When $d > 2$, $\boldsymbol{B}$ would never be materialized; instead we repeatedly apply Equation 7 to get a representation of $\boldsymbol{B}$ that can be used to compute matrix-vector products.

Assuming for simplicity that $\boldsymbol{A}_i \in \mathbb{R}^{n \times n}$ for all i, the space and time complexity of computing $\mathsf{Multiply}(\mathbb{A}, \boldsymbol{x})$ using this procedure is $O(n^d)$ and $O(dn^{d+1})$ respectively, where n^d is the size of the data vector. Using an explicit matrix representation would require $O(n^{2d})$ time and space.

Related techniques allow RECONSTRUCT to be accelerated because $\mathsf{LstSqr}(\mathbb{A}, \boldsymbol{y})$ can be defined as $\mathsf{Multiply}(\mathbb{A}^+, \boldsymbol{y})$ where $\mathbb{A}^+$ is the pseudo inverse of $\mathbb{A}$. For strategies produced by $\mathsf{OPT}_{\otimes}$ and $\mathsf{OPT}_{\mathsf{M}}$, the pseudo inverse can be computed efficiently in implicit form. For $\mathsf{OPT}_{\otimes}$, we use the identity in Section 4.4: $(\boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d)^+ = \boldsymbol{A}_1^+ \otimes \cdots \otimes \boldsymbol{A}_d^+$. For $\mathsf{OPT}_{\mathsf{M}}$, the pseudo inverse of the strategy $\mathbb{M}(\boldsymbol{\theta})$ is $\mathbb{M}^+ = (\mathbb{M}^T\mathbb{M})^+\mathbb{M}^T$, where $\mathbb{M}^T$ is a (transposed) union of Kronecker products, and $(\mathbb{M}^T\mathbb{M})^+$ is a sum of Kronecker products. Thus, we can define an efficient routine for LstSqr for these strategies. Unfortunately, we are not aware of an efficient method for computing the pseudo inverse of a strategy produced by OPT_{+}, but we can still perform inference by using an iterative algorithm like LSMR [14] to solve the least squares problem. This algorithm only requires as input a routine for computing matrix-vector products on $\mathbb{A}$ and $\mathbb{A}^T$, which we can do efficiently using Multiply.

7.3 Privacy statement

We conclude with a statement of the privacy of HDMM. The ImpVec and $\mathsf{OPT}_{\mathsf{HDMM}}$ steps of HDMM do not use the input $\boldsymbol{x}$. The Laplace Mechanism is used to compute $\tilde{\boldsymbol{x}}$ from $\boldsymbol{x}$ and the privacy of HDMM follows from privacy properties of the Laplace Mechanism [11] and the well-known post-processing theorem [12].

THEOREM 7 (PRIVACY). *The* HDMM *algorithm is* ϵ*-differentially private.*

8. EXPERIMENTS

In this section we evaluate the accuracy and scalability of HDMM. We begin with details on the experimental setup. In Section 8.2 we perform a comprehensive comparison of HDMM with competing algorithms, showing that it consistently offers lower error and works in a significantly broader range of scenarios than other algorithms. In Section 8.3, we study the scalability of HDMM compared with the subset of

methods that perform search over a general strategy space, showing the HDMM scales more favorably than competitors and can efficiently support high-dimensional cases.

8.1 Experimental setup

Implementation Details Our Python implementation uses the L-BFGS-B algorithm [5], as implemented in `scipy.optimize`, as the solver for all optimization routines. Scalability experiments were done on a 4-core Intel i7 3.6GHz processor with 16GB of RAM. The parameter p, controlling the size of p-Identity strategies for OPT_0, is set as described in Section 7.1. In experiments not shown, we varied p and found that any choice between $\frac{n}{32}$ and $\frac{n}{8}$ results in nearly the same accuracy. In experiments, the number of restarts (S in Algorithm 2) is set to 25. We observed that the distribution of local minima across different random restarts was fairly concentrated, suggesting that far fewer than 25 restarts may be sufficient in practice.

Competing techniques We compare HDMM against a variety of techniques from the literature. Some algorithms are specialized to particular settings and we indicate that below.

We consider two baseline algorithms: the Laplace Mechanism (LM) and Identity. LM adds noise directly to each workload query (scaled to the sensitivity) [21,22]. Identity adds noise to the entries of the data vector, then uses the noisy data vector to answer the workload queries. Identity and LM work in both low and high dimensions, on any input workload.

We also consider the two select-measure-reconstruct methods with general search spaces: The Matrix Mechanism (MM) [28] and the Low Rank Mechanism (LRM) [47]. Both of these attempt to find a strategy that minimizes total squared error on the workload queries.

In addition to these general-purpose algorithms, we consider a number of other algorithms designed for low-dimensions and specialized for specific workload classes. These include Privelet [41], HB [34], Quadtree [8], and GreedyH [24]. These algorithms are all designed to accurately answer range queries: Privelet uses a Haar wavelet as the strategy, HB uses a hierarchical strategy with branching factor that adapts to the domain size, GreedyH uses a weighted hierarchical strategy, and Quadtree uses a generalization of the hierarchical strategy to two dimensions. Of the above methods, Privelet, HB, and Quadtree tend to be quite scalable but have limited search spaces. GreedyH solves a non-trivial optimization problem making it less scalable. For workloads consisting solely of marginals, we also compare against DataCube [10]. DataCube accepts as input a workload of marginals, and returns a strategy consisting of a different set of marginals that adapts to the input through a greedy heuristic.

All of the algorithms described so far, with the exception of LM, are members of the select-measure-reconstruct paradigm. We also consider two state-of-the-art algorithms outside of this class: DAWA [24] and PrivBayes [48]. In 1- or 2-dimensions, the DAWA algorithm uses some of its privacy budget to detect approximately uniform regions, compresses the domain, reformulates the workload, and then uses the remainder of its privacy budget to run the GreedyH algorithm described above. PrivBayes is suitable for multi-dimensional datasets. PrivBayes first privately fits a Bayesian network on the data and then generates a synthetic dataset by drawing samples from the Bayesian network. The synthetic data can then be used to answer the workload. Note that both DAWA and PrivBayes have error rates that depend on the input data.

Datasets We consider five datasets, covering low and high dimensional cases. Most of the algorithms we consider have error rates that only depend on the schema and not the dataset instance. The different schemas we consider have a large impact on the workloads that can be defined over the data and runtime complexity of algorithms.

For 1D and 2D cases, we use representative datasets from the DPBench study [18], *Patent* and BeijingTaxiE (which we call *Taxi*). For higher dimensional cases, we use three datasets, each derived from different Census products. *CPH* (short for Census of Population and Housing) is the dataset used as a running example throughout the paper and is described in Section 2. *Adult* is a dataset from the UCI machine learning dataset repository [9] with five discrete attributes for age, education, race, sex, and hours-per-week. *CPS* is a dataset released in the March 2000 Population Survey conducted by the Census [1]; it has five discrete attributes for income, age, marital status, race, and sex.

For scalability experiments we use synthetic datasets, allowing us to systematically vary the dimensionality and attribute domain sizes. The runtime of HDMM, and the other algorithms we compare against, only depends on the domain size and dimensionality of the data, and not the contents of the data vector, so we use an all-zero data vector.

Workloads For the *CPH* dataset, we use the *SF1* and *SF1+* workloads that were introduced in Section 2 and used as a motivating use case throughout the paper. For the other datasets, we selected workloads that we believe are representative of typical data analyst interactions. We also synthesized a few workloads that help illustrate differences in algorithm behavior.

For 1-dimensional datasets, we use three workloads based on range queries: *Prefix 1D*, *Width 32 Range*, and *Permuted Range*. *Prefix 1D* is P, as described in Section 3.3, and serves as a compact proxy for all range queries. The *Width 32 Range* workload contains all range queries that sum 32 contiguous elements of the domain (i.e., it omits small ranges). *Permuted Range* is a workload consisting of all range queries right-multiplied by a random permutation matrix to randomly shuffle the elements of the domain. This synthesized workload serves to evaluate whether algorithms can "recover" the obscured range query workload.

For 2-dimensional datasets, we use workloads *Prefix 2D* and *Prefix Identity*. The *Prefix 2D* workload is just the product workload $P \times P$. The *Prefix Identity* workload is a union of two products: $P \times I$ and $I \times P$.

For higher dimensional datasets, we use a variety of workloads. We consider multiple workloads based on marginals. *All Marginals* contains queries for the set of 2^d marginals (for a dataset of dimension d); *2-way Marginals* contains queries for the $\binom{d}{2}$ 2-way marginals; and *3-way Marginals* contains queries for the $\binom{d}{3}$ 3-way marginals. We also consider a variation on marginals in which range queries are included for numerical attributes (like income and age): *All Range-Marginals* is a marginals-like workload, but the Identity subworkloads are replaced by range query workloads on the numerical attributes, and *2-way Range-Marginals* is a

Table 3: Error ratios of various algorithms on low and high dimension datasets/workloads with $\epsilon = 1.0$. Algorithms labeled - are not applicable for the given configuration; algorithms labeled * are not scalable to the given configuration.

Configuration			General-Purpose Algorithms					Low-D Range Query Algorithms					High-D Algorithms	
Dataset	Domain / Dimensions	Workload	Identity	LM	MM	LRM	HDMM	Privelet	HB	Quadtree	GreedyH	DAWA	DataCube	PrivBayes
Patent	**1024**	*Width 32 Range*	**1.25**	7.06	*	3.21	***1.0***	2.59	1.48	-	**1.25**	2.45	-	-
		Prefix 1D	3.34	151	*	2.44	***1.0***	1.80	**1.34**	-	1.49	2.96	-	-
		Permuted Range	2.36	877000	*	*	***1.0***	10.57	3.35	-	**2.16**	*	-	-
Taxi	$\mathbf{256 \times 256}$	*Prefix Identity*	**1.44**	65.0	*	*	***1.0***	6.11	4.05	4.71	*	*	-	-
		Prefix 2D	4.75	2422	*	*	***1.0***	3.14	2.03	**1.95**	*	*	-	-
CPH	$\mathbf{2 \times 2 \times 64 \times 17 \times 115 \times 51}$	*SF1*	**3.07**	9.32	*	*	***1.0***	-	-	-	-	-	-	66700
		SF1+	**3.16**	13.7	*	*	***1.0***	-	-	-	-	-	-	6930
Adult	$\mathbf{75 \times 16 \times 5 \times 2 \times 20}$	*All Marginals*	**1.38**	11.2	*	*	***1.0***	-	-	-	-	-	4.57	20.5
		2-way Marginals	5.30	2.11	*	*	***1.0***	-	-	-	-	-	**2.01**	155
CPS	$\mathbf{100 \times 50 \times 7 \times 4 \times 2}$	*All Range-Marginals*	**1.49**	421000	*	*	***1.0***	-	-	-	-	-	-	4.74
		2-way Range-Marginals	**5.79**	53200	*	*	***1.0***	-	-	-	-	-	-	24.8

subset of the previous workload that only contains queries over two dimensions at a time. *Prefix 3D* is the set of prefix queries along three dimensions ($P \times P \times P$); and *All 3-way Ranges* is the set of all 3-way range queries.

Error measures To compare HDMM to other algorithms in terms of accuracy, we report *error ratios*. Recall from Section 3 that $Err(W, \mathcal{K})$ is the expected total squared error of algorithm $\mathcal{K}$ for workload W, and that for algorithms within the select-measure-reconstruct paradigm, this quantity can be computed in closed form (Definition 7) and is independent of the input data. Define the *error ratio* between $\mathcal{K}_{other}$ and HDMM on workload W as $Ratio(W, \mathcal{K}_{other}) = \sqrt{\frac{Err(W,\mathcal{K}_{other})}{Err(W,\text{HDMM})}}$. Whenever possible, we report analytically computed error ratios (which hold for all settings of ϵ.) For data-dependent algorithms (DAWA and PrivBayes), the expected error depends on the input data and ϵ, so these are stated in our comparisons. There is no closed form expression for expected error of a data-dependent algorithm, so we estimate it using average error across 25 random trials.

8.2 Accuracy comparison

To assess the accuracy of HDMM, we considered 11 competing algorithms and a total of 11 workloads, defined over two low-dimensional datasets and three high-dimensional datasets. Our goal was to empirically compare error of all algorithms in all settings. However, some algorithms are not defined for some of the datasets and workloads; these cases are labeled with $-$ in Table 3. For example, a number of algorithms were designed for low-dimensions, while others were designed for high dimensions. In addition, there are algorithms that are defined for a given dataset/workload, but were infeasible to run; these cases are labeled with $*$ in Table 3. For example, in theory MM is applicable to any workload, but it is infeasible to run for the domain sizes we considered. LRM is also applicable to any workload but is only feasible on medium-sized or smaller domains where the workload and strategy can be represented as a dense matrix. Overall, HDMM and the simple baseline methods (LM and Identity) are the only algorithms general and scalable enough to run in all target settings.

Findings Table 3 summarizes the results, reported as error ratios to HDMM (so that HDMM is always 1.0). It shows that HDMM is *never outperformed by any competing method* and offers significant improvements, often at least a factor of two and sometimes an order of magnitude.

Even when the workload is low-dimensional range queries, and we compare against a collection of algorithms designed specifically for this workload (e.g. HB, Privelet, GreedyH), HDMM is 1.34 times better than the best algorithm, HB. On the *Permuted Range* workload, only HDMM offers acceptable utility, since the other algorithms are specifically designed for (unpermuted) range queries, while HDMM adapts to a broader class of workloads.

Overall, we see that some algorithms approach the error offered by HDMM (ratios close to 1), but, importantly, only for some experimental configurations. To highlight this, we use bold in the table to indicate the *second best* error rate, after HDMM. We find that, depending on the workload and domain size, the second best error rate is achieved by a broad set of algorithms: Identity, HB, Quadtree, GreedyH are all second-best performers for some workload. This shows that some competing algorithms have specialized capabilities that allow them to perform well in some settings. In contrast, HDMM outperforms across all settings, improving error and simplifying the number of algorithms that must be implemented for deployment.

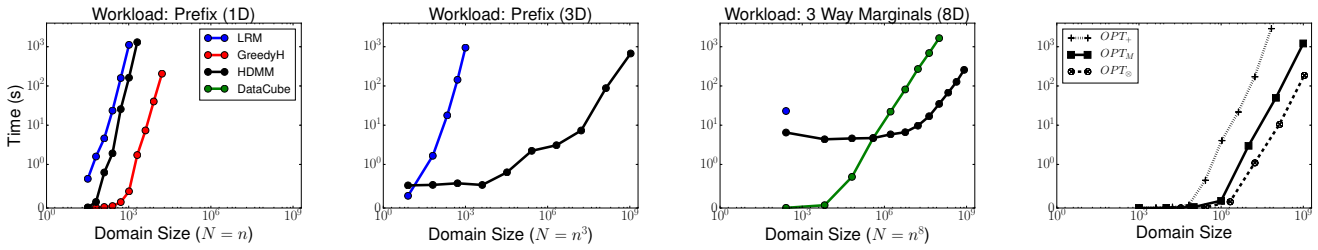
In high dimensions, there are fewer competitors, and Identity is generally the best alternative to HDMM, but the magnitude of improvement by HDMM can be as large as 5.79. Included in Table 3 are two data-dependent algorithms. DAWA is defined only for 1D and 2D, but in fact timed out for some low-dimensional workloads. PrivBayes is designed for high-dimensional data, but does not offer competitive accuracy for these datasets. Note that the error rates for these methods may differ on different datasets (and for different ϵ values).

8.3 Scalability comparison

Next we evaluate the scalability of HDMM by systematically varying the domain size and dimensionality. We assume throughout that each dimension has a fixed size, n, so that d-dimensional data has a total domain size of $N = n^d$. We ran each algorithm on increasingly larger datasets until it exhausted memory or reached a 30-minute timeout.

We compare against the other general-purpose algorithms: LRM, GreedyH, and DataCube. We omit MM because it cannot run at these domain sizes and Identity, Privelet, HB, and Quadtree because (as shown above) they offer sub-optimal error and specialize to narrow workload classes.

Figure 1a shows the scalability of LRM, GreedyH, and HDMM on the *Prefix 1D* workload. (DataCube is listed as not applicable (N/A) because it is not defined on non-marginal workloads.) Since all three of these algorithms require as input an explicitly represented workload in dense matrix form, they are unable to scale beyond $N \approx 10^4$. On this 1D workload, HDMM runs a single instance of OPT_0, which can be expensive as the domain size grows. HDMM is more scalable than LRM, but less scalable then GreedyH. (Recall HDMM is run with 25 random restarts; in results not shown, we lower the number of restarts to 1. At this



(a) N/A: DataCube **(b)** N/A: GreedyH, DataCube **(c)** N/A: GreedyH **(d)** Measure+reconstruct

Figure 1: (a)-(c) Runtime comparison on synthetic datasets of increasing domain size (workload and dimensionality varies by plot); (d) Runtime of HDMM's measure+reconstruct phase on strategies selected by its $\mathsf{OPT}_\otimes$, OPT_+ and OPT_M subroutines.

setting, HDMM still achieves higher accuracy than other methods yet achieves lower runtime than GreedyH.)

Figure 1b shows the scalability of LRM and HDMM on the *Prefix 3D* workload. (GreedyH is only applicable in 1D; DataCube remains inapplicable.) The scalability of LRM is roughly the same in 3D as it is in 1D because it is determined primarily by the total domain size. HDMM is far more scalable in 3D however, because it solves three smaller optimization problems (for $\mathsf{OPT}_\otimes$) rather than one large problem. The main bottleneck for HDMM is measure and reconstruct, not strategy optimization.

Figure 1c shows the scalability of DataCube and HDMM on the *3-way Marginals* workload (again, GreedyH does not apply to high dimensions). Both DataCube and HDMM scale well, with DataCube scaling to domains as large as $N \approx 10^8$ and HDMM scaling to $N \approx 10^9$. On small domains, DataCube is faster than HDMM, due to HDMM's higher up front optimization cost (25 random restarts and multiple optimization programs). As the domain size grows, measure and reconstruct becomes the bottleneck for both algorithms. A single data point is shown for LRM because it did not finish on larger domain sizes ($N \geq 3^8$).

In summary, HDMM is most scalable in the multi-dimensional setting, where our implicit representations speed up strategy selection. The main bottleneck is the measure and reconstruct steps, which we now examine more closely.

Unlike strategy selection, the cost of measure+reconstruct primarily depends on the total domain size (since the data vector must be explicitly represented). Figure 1d shows the runtime of measure+reconstruct for synthetic datasets of varying domain sizes. Since we designed specialized algorithms for each strategy type produced by HDMM (as described in Section 7.2), there is one line for each strategy selection subroutine. On strategies produced by $\mathsf{OPT}_\otimes$ and OPT_M, measure+reconstruct scales to domains as large as $N \approx 10^9$—at which point, the data vector is 4 GB in size (assuming 4 byte floats). OPT_+ does not scale as well ($N \approx 10^8$). This is because computing the pseudo-inverse for OPT_+ requires iterative methods, whereas $\mathsf{OPT}_\otimes$ and OPT_M have closed-form expressions that we exploit.

9. DISCUSSION AND CONCLUSIONS

HDMM is a general and scalable method for privately answering collections of counting queries over high-dimensional data. Because HDMM provides state-of-the-art error rates in both low- and high-dimensions, and fully automated strategy selection, we believe it will be broadly useful to algorithm designers.

HDMM is capable of running on multi-dimensional datasets with very large domains. This is primarily enabled by our implicit workload representation in terms of Kronecker products, and our optimization routines for strategy selection that exploit this implicit representation. We also exploit the structure of the strategies produced by the optimization to efficiently solve the least squares problem.

HDMM is limited to cases for which it is possible to materialize and manipulate the data vector. Since we have only investigated a centralized, single-node implementation, it is possible HDMM could be scaled to larger data vectors, especially since we have shown that strategy selection is not the bottleneck. Recent work has shown that standard operations on large matrices can be parallelized [38], however the decomposed structure of our strategies should lead to even faster specialized parallel solutions. Ultimately, for very large domains, factoring the data (as PrivBayes does) may be unavoidable. HDMM still has a role to play, however, since it can be used to optimize queries over the factored subsets of the data.

As noted previously, HDMM optimizes for absolute error and is not applicable to optimizing relative error, which is data dependent. Nevertheless, by weighting the workload queries (e.g. inversely with their L_1-norm) we can approximately optimize relative error, at least for datasets whose data vectors are close to uniform. This approach could be extended to reflect a user's assumptions or guesses about the input data. Future work could also integrate HDMM measurement with techniques like iReduct [39] which perform adaptive measurement to target relative error.

While HDMM produces the best known strategies for a variety of workloads, we do not know how close to optimal its solutions are. There are asymptotic lower bounds on error in the literature [15, 17, 31], but it is not clear how to use them on a concrete case given hidden constant factors. Li et al [27] provided a precise lower bound in terms of the spectral properties of $\boldsymbol{W}$, but it is not clear how to compute it on our large workload matrices and it is often a very loose lower bound under ϵ-differential privacy.

Acknowledgements: This work was supported by the National Science Foundation under grants 1253327, 1408982, 1409125, 1443014, 1421325, and 1409143; and by DARPA and SPAWAR under contract N66001-15-C-4067. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright notation thereon. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

10. REFERENCES

[1] Current population survey. Available at: https://www2.census.gov/programs-surveys/cps/techdocs/cpsmar00.pdf, March 2000.

[2] G. Ács, C. Castelluccia, and R. Chen. Differentially private histogram publishing through lossy compression. In *ICDM*, pages 1–10, 2012.

[3] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. 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*, pages 273–282. ACM, 2007.

[4] A. Bhaskara, D. Dadush, R. Krishnaswamy, and K. Talwar. Unconditional differentially private mechanisms for linear queries. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 1269–1284, New York, NY, USA, 2012. ACM.

[5] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[6] 2010 Census Summary File 1, Census of Population and Housing. Available at `https://www.census.gov/prod/cen2010/doc/sf1.pdf`, 2012.

[7] Census scientific advisory committee meeting. www.census.gov/about/cac/sac/meetings/2017-09-meeting.html, September 2017.

[8] G. Cormode, C. Procopiuc, D. Srivastava, E. Shen, and T. Yu. Differentially private spatial decompositions. In *Data engineering (ICDE), 2012 IEEE 28th international conference on*, pages 20–31. IEEE, 2012.

[9] D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017.

[10] B. Ding, M. Winslett, J. Han, and Z. Li. Differentially private data cubes: optimizing noise sources and consistency. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 217–228. ACM, 2011.

[11] C. Dwork, F. M. K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[12] C. Dwork and A. Roth. *The Algorithmic Foundations of Differential Privacy*. Found. and Trends in Theoretical Computer Science, 2014.

[13] H. Ebadi, D. Sands, and G. Schneider. Differential privacy: Now it's getting personal. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 69–81, New York, NY, USA, 2015. ACM.

[14] D. C.-L. Fong and M. Saunders. Lsmr: An iterative algorithm for sparse least-squares problems. *SIAM Journal on Scientific Computing*, 33(5):2950–2971, 2011.

[15] A. Gupta, A. Roth, and J. Ullman. Iterative constructions and private data release. In *Proceedings of the 9th International Conference on Theory of Cryptography*, TCC'12, pages 339–356, Berlin, Heidelberg, 2012. Springer-Verlag.

[16] S. Haney, A. Machanavajjhala, M. Kutzbach, M. Graham, J. Abowd, and L. Vilhuber. Utility cost of formal privacy for releasing national employer-employee statistics. In *ACM SIGMOD*, 2017.

[17] M. Hardt and K. Talwar. On the geometry of differential privacy. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 705–714, New York, NY, USA, 2010. ACM.

[18] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang. Principled evaluation of differentially private algorithms using dpbench. In *Proceedings of the 2016 International Conference on Management of Data*, pages 139–154. ACM, 2016.

[19] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 3(1-2):1021–1032, 2010.

[20] HCUPnet: Healthcare Cost and Utilization Project. Available at `https://hcupnet.ahrq.gov/`.

[21] A. Inan, M. E. Gursoy, E. Esmerdag, and Y. Saygin. Graph-based modelling of query sets for differential privacy. In *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*, SSDBM '16, pages 3:1–3:10, New York, NY, USA, 2016. ACM.

[22] N. Johnson, J. P. Near, and D. Song. Towards practical differential privacy for sql queries. *PVLDB*, 11(5):526–539, 2018.

[23] D. Kifer and A. Machanavajjhala. Pufferfish: A framework for mathematical privacy definitions. *ACM Transactions on Database Systems (TODS)*, 39(1):1–36, 2014.

[24] C. Li, M. Hay, G. Miklau, and Y. Wang. A data-and workload-aware algorithm for range queries under differential privacy. *PVLDB*, 7(5):341–352, 2014.

[25] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 123–134. ACM, 2010.

[26] C. Li and G. Miklau. An adaptive mechanism for accurate query answering under differential privacy. *PVLDB*, 5(6):514–525, 2012.

[27] C. Li and G. Miklau. Optimal error of query sets under the differentially-private matrix mechanism. In *ICDT*, 2013.

[28] C. Li, G. Miklau, M. Hay, A. McGregor, and V. Rastogi. The matrix mechanism: optimizing linear counting queries under differential privacy. *The VLDB Journal*, 24(6):757–781, 2015.

[29] A. Machanavajjhala, D. Kifer, J. M. Abowd, J. Gehrke, and L. Vilhuber. Privacy: Theory meets practice on the map. In *ICDE*, pages 277–286, 2008.

[30] F. D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 19–30, New York, NY, USA, 2009. ACM.

[31] A. Nikolov, K. Talwar, and L. Zhang. The geometry of differential privacy: The sparse and approximate cases. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 351–360, New York, NY, USA, 2013. ACM.

[32] OnTheMap Web Tool. Available at `http://onthemap.ces.census.gov/`.

[33] W. Qardaji, W. Yang, and N. Li. Differentially private grids for geospatial data. In *Intl. Conference on Data Engineering (ICDE)*, pages 757–768. IEEE, 2013.

[34] W. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *PVLDB*, 6(14):1954–1965, 2013.

[35] W. Qardaji, W. Yang, and N. Li. Priview: practical differentially private release of marginal contingency tables. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1435–1446. ACM, 2014.

[36] J. Vaidya, B. Shafiq, X. Jiang, and L. Ohno-Machado. Identifying inference attacks against healthcare data repositories. *AMIA Jt Summits Transl Sci Proc*, 2013, 2013.

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

[38] J. Xiang, H. Meng, and A. Aboulnaga. Scalable matrix inversion using mapreduce. In *High-performance Parallel and Distributed Computing*, HPDC '14, 2014.

[39] X. Xiao, G. Bender, M. Hay, and J. Gehrke. ireduct: Differential privacy with reduced relative errors. In *SIGMOD*, 2011.

[40] X. Xiao and Y. Tao. Output perturbation with query relaxation. *PVLDB*, 1(1):857–869, Aug. 2008.

[41] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE Transactions on Knowledge and Data Engineering*, 23(8):1200–1214, 2011.

[42] Y. Xiao, L. Xiong, L. Fan, S. Goryczka, and H. Li. DPCube: Differentially private histogram release through multidimensional partitioning. *Transactions of Data Privacy*, 7(3), 2014.

[43] J. Xu, Z. Zhang, X. Xiao, Y. Yang, and G. Yu. Differentially private histogram publication. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 32–43, 2012.

[44] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett. Differentially private histogram publication. *The VLDB Journal*, pages 1–26, 2013.

[45] G. Yaroslavtsev, G. Cormode, C. M. Procopiuc, and D. Srivastava. Accurate and efficient private release of datacubes and contingency tables. In *ICDE*, 2013.

[46] G. Yuan, Y. Yang, Z. Zhang, and Z. Hao. 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*, pages 2005–2014. ACM, 2016.

[47] G. Yuan, Z. Zhang, M. Winslett, X. Xiao, Y. Yang, and Z. Hao. Low-rank mechanism: optimizing batch queries under differential privacy. *PVLDB*, 5(11):1352–1363, 2012.

[48] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems (TODS)*, 42(4):25, 2017.

[49] J. Zhang, X. Xiao, and X. Xie. Privtree: A differentially private algorithm for hierarchical decompositions. In *SIGMOD*, 2016.

[50] X. Zhang, R. Chen, J. Xu, X. Meng, and Y. Xie. Towards accurate histogram publication under differential privacy. In *SDM*, 2014.