Convex Optimization for Linear Query Processing under Approximate Differential Privacy

Ganzhao Yuan[1] Yin Yang[2] Zhenjie Zhang[3] Zhifeng Hao[4]

[1]School of Mathematics
South China University of Technology
yuanganzhao@gmail.com

[2]College of Science and Engineering
Hamad Bin Khalifa University
yyang@qf.org.qa

[3]Advanced Digital Sciences Center
Illinois at Singapore Pte. Ltd.
zhenjie@adsc.com.sg

[4]School of Mathematics and Big Data
Foshan University
mazfhao@scut.edu.cn

ABSTRACT

Differential privacy enables organizations to collect accurate aggregates over sensitive data with strong, rigorous guarantees on individuals' privacy. Previous work has found that under differential privacy, computing multiple correlated aggregates as a batch, using an appropriate *strategy*, may yield higher accuracy than computing each of them independently. However, finding the best strategy that maximizes result accuracy is non-trivial, as it involves solving a complex constrained optimization program that appears to be non-convex. Hence, in the past much effort has been devoted in solving this non-convex optimization program. Existing approaches include various sophisticated heuristics and expensive numerical solutions. None of them, however, guarantees to find the optimal solution of this optimization problem.

This paper points out that under (ϵ, δ)-differential privacy, the optimal solution of the above constrained optimization problem in search of a suitable strategy can be found, rather surprisingly, by solving a simple and elegant convex optimization program. Then, we propose an efficient algorithm based on Newton's method, which we prove to always converge to the optimal solution with linear global convergence rate and quadratic local convergence rate. Empirical evaluations demonstrate the accuracy and efficiency of the proposed solution.

1. INTRODUCTION

Differential privacy [4, 2] is a strong and rigorous privacy protection model that is known for its generality, robustness and effectiveness. It is used, for example, in the ubiquitous Google Chrome browser [6]. The main idea is to publish randomized aggregate information over sensitive data, with the guarantee that the adversary cannot infer with high confidence the presence or absence of any individual in the dataset from the released aggregates. An important goal in the design of differentially private methods is to maximize the accuracy of the published noisy aggregates with respect to their exact values.

KDD '16, August 13-17, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-4232-2/16/08...$15.00
DOI: http://dx.doi.org/10.1145/2939672.2939818

Besides optimizing for specific types of aggregates, an important generic methodology for improving the overall accuracy of the released aggregates under differential privacy is *batch processing*, first proposed in [12]. Specifically, batch processing exploits the correlations between multiple queries, so that answering the batch as a whole can lead to higher overall accuracy than answering each query individually. For example, if one aggregate query Q1 (e.g., the total population of New York State and New Jersey) can be expressed as the sum of two other queries (the population of New York and New Jersey, respectively), i.e., Q1 = Q2 + Q3, then we can simply answer Q1 by adding up the noisy answers of Q2 and Q3. Intuitively, answering two queries instead of three reduces the amount of random perturbations required to satisfy differential privacy, leading to higher overall accuracy for the batch as a whole [12, 23]. In this paper, we focus on answering linear aggregate queries under differential privacy. Given a batch of linear aggregate queries (called the *workload*), we aim to improve their overall accuracy by answering a different set of queries (called the *strategy*) under differential privacy, and combining their results to obtain the answers to the original workload aggregates.

As shown in [12, 13, 23, 24, 10], different strategy queries lead to different overall accuracy for the original workload. Hence, an important problem in batch processing under differential privacy is to find a suitable strategy that leads to the highest accuracy. Such a strategy can be rather complex, rendering manual construction and brute-force search infeasible [23, 24]. On the other hand, the problem of strategy searching can be formulated into a constrained optimization program, and it suffices to find the optimal solution of this program. However, as we show later in Section 2, the program appears to be non-convex; hence, solving it is rather challenging. To the best of our knowledge, existing approaches resort to either heuristics or complex, expensive and unstable numerical methods. To our knowledge, no existing solutions guarantee to find the optimal solution.

This paper points out that under the (ϵ, δ)-differential privacy definition (also called approximate differential privacy, explained in Section 2), the constrained optimization program for finding the best strategy queries can be re-formulated into a simple and elegant convex optimization program. Note that although the formulation itself is simple, its derivation is rather complicated and non-trivial. Based on this new formulation, we propose the first polynomial solution COA that *guarantees to find the optimal solution* to the original constrained optimization problem in search of a suitable strategy for processing a batch of arbitrary linear aggregate queries under approximate differential privacy. COA is based on Newton's method and it utilizes various non-trivial properties of the problem. We show that COA achieves globally linear and locally quadratic convergence rate. Extensive experiments confirm the effectiveness and efficiency of the proposed method.

The rest of the paper is organized as follows. Section 2 provides necessary background on differential privacy and overviews related work. Section 3 presents our convex programming formulation for batch linear aggregate processing under approximate differential privacy. Section 4 describes the proposed solution COA. Section 5 contains a thorough set of experiments. Section 6 concludes the paper with directions for future work. In this paper, boldfaced lowercase letters denote vectors and uppercase letters denote real-valued matrices. We summarize the frequent notations in Table 1.

2. BACKGROUND

Using the notations summarized in Table 1, a common definition of differential privacy is (ϵ, δ)-differential privacy [4], as follows:

Definition 1. *Two databases D and D' are neighboring iff they differ by at most one tuple. A randomized algorithm $\mathcal{M}$ satisfies (ϵ, δ)-differential privacy iff for any two neighboring databases D and D' and any measurable output $\mathbb{S}$ in the range of $\mathcal{M}$, we have*

$$\Pr[\mathcal{M}(D) \in \mathbb{S}] \leq e^{\epsilon} \cdot \Pr[\mathcal{M}(D') \in \mathbb{S}] + \delta.$$

When $\delta = 0$, the above definition reduces to another popular definition: ϵ-differential privacy (also called "exact differential privacy"). This work focuses on the case where $\delta > 0$, which is sometimes called approximate differential privacy. Usually, δ is set to a value smaller than $\frac{1}{|D|}$, where $|D|$ is the number of records in the dataset D. Both exact and approximate definitions of differential privacy provide strong and rigorous privacy protection to the users. Given the output of a differentially private mechanism, the adversary cannot infer with high confidence (controlled by parameters ϵ and δ) whether the original database is D or any of its neighbors D', which differ from D by one record, meaning that each user can plausibly deny the presence of her tuple. An approximately differentially private mechanism can be understood as satisfying exact differential privacy with a certain probability controlled by parameter δ. Hence, it is a more relaxed definition which is particularly useful when the exact definition is overly strict for an application, leading to poor result utility.

One basic mechanism for enforcing approximate differential privacy is the Gaussian mechanism [3], which injects Gaussian noise to the query results calibrated to the ℓ_2 sensitivity of the queries. Note that the Gaussian mechanism cannot be applied to exact differential privacy. Since the proposed method is based on the Gaussian mechanism, it is limited to query processing under approximate differential privacy as well. Specifically, for any two neighbor databases D and D', the ℓ_2 sensitivity $\Theta(Q)$ of a query set Q is defined as $\Theta(Q) = \max_{D,D'} \|Q(D), Q(D')\|_2$. Given a database D and a query set Q, the Gaussian mechanism outputs a random result that follows the Gaussian distribution with mean $Q(D)$ and magnitude $\sigma = \Theta(Q)/h(\epsilon, \delta)$, where $h(\epsilon, \delta) = \epsilon/\sqrt{2\ln(2/\delta)}$.

This paper focuses on answering a batch of m linear aggregate queries, $Q = \{q_1, q_2, \ldots, q_m\}$, each of which is a linear combination of the unit aggregates of the input database D. For simplicity, in the following we assume that each unit aggregate is a simple count, which has an ℓ_2 sensitivity of 1. Other types of aggregates can be handled by adjusting the sensitivity accordingly. The query set Q can be represented by a *workload matrix* $\mathbf{W} \in \mathbb{R}^{m\times n}$ with m rows and n columns. Each entry $\mathbf{W}_{ij}$ in $\mathbf{W}$ is the weight in query

Table 1: Summary of notations.

Symbol	Definition
$\mathbf{W}$	$\mathbf{W} \in \mathbb{R}^{m\times n}$, Workload matrix
m	Number of queries (i.e., rows) in $\mathbf{W}$
n	Unit counts (i.e., columns) in $\mathbf{W}$
$\mathbf{V}$	$\mathbf{V} \in \mathbb{R}^{n\times n}$, Covariance matrix of $\mathbf{W}$
$\mathbf{X}$	$\mathbf{X} \in \mathbb{R}^{n\times n}$, Solution matrix
$\mathbf{A}$	$\mathbf{A} \in \mathbb{R}^{p\times n}$, Strategy matrix
$\mathbf{A}^{\dagger}$	$\mathbf{A}^{\dagger} \in \mathbb{R}^{n\times p}$, pseudo-inverse of matrix $\mathbf{A}$
$vec(\mathbf{X})$	$vec(\mathbf{X}) \in \mathbb{R}^{n^2\times 1}$, Vectorized listing of $\mathbf{X}$
$mat(\mathbf{x})$	$mat(\mathbf{x}) \in \mathbb{R}^{n\times n}$, Convert $\mathbf{x} \in \mathbb{R}^{n^2\times 1}$ into a matrix
$F(\mathbf{X})$	$F(\mathbf{X}) \in \mathbb{R}$, Objective value of $\mathbf{X}$
$G(\mathbf{X})$	$G(\mathbf{X}) \in \mathbb{R}^{n\times n}$, Gradient matrix of $\mathbf{X}$
$H(\mathbf{X})$	$H(\mathbf{X}) \in \mathbb{R}^{n^2\times n^2}$, Hessian matrix of $\mathbf{X}$
$\mathcal{H}_{\mathbf{X}}(\mathbf{D})$	$\mathcal{H}_{\mathbf{X}}(\mathbf{D}) \in \mathbb{R}^{n\times n}$, Equivalent to $mat(H(\mathbf{X})vec(\mathbf{D}))$
1/**0**	All-one column vector/All-zero column vector
$\mathbf{I}$	Identity matrix
$\mathbf{X} \succeq 0$	Matrix $\mathbf{X}$ is positive semidefinite
$\mathbf{X} \succ 0$	Matrix $\mathbf{X}$ is positive definite
$\boldsymbol{\lambda}(\mathbf{X})$	Eigenvalue of $\mathbf{X}$ (increasing order)
$diag(\mathbf{x})$	Diagonal matrix with $\mathbf{x}$ as the main diagonal entries
$diag(\mathbf{X})$	Column vector formed from the main diagonal of $\mathbf{X}$
$\|\mathbf{X}\|$	Spectral norm: the largest eigenvalue of $\mathbf{X}$
$\chi(\mathbf{X})$	Smallest eigenvalue of $\mathbf{X}$
$tr(\mathbf{X})$	Sum of the elements on the main diagonal $\mathbf{X}$
$\langle \mathbf{X}, \mathbf{Y} \rangle$	Euclidean inner product, i.e., $\langle \mathbf{X}, \mathbf{Y} \rangle = \sum_{ij} \mathbf{X}_{ij}\mathbf{Y}_{ij}$
$\mathbf{X} \otimes \mathbf{Y}$	Kronecker product of $\mathbf{X}$ and $\mathbf{Y}$
$\mathbf{X} \odot \mathbf{Y}$	Hadamard (a.k.a. entry-wise) product of $\mathbf{X}$ and $\mathbf{Y}$
$\|\mathbf{X}\|_*$	Nuclear norm: sum of the singular values of matrix $\mathbf{X}$
$\|\mathbf{X}\|_F$	Frobenius norm: $(\sum_{ij} \mathbf{X}_{ij}^2)^{1/2}$
$\|\mathbf{X}\|_{\mathbf{N}}$	Generalized vector norm: $\|\mathbf{X}\|_{\mathbf{N}} = (vec(\mathbf{X})^T\mathbf{N}vec(\mathbf{X}))^{1/2}$

q_i on the j-th unit count $\mathbf{x}_j$. Since we do not use any other information of the input database D besides the unit counts, in the following we abuse the notation by using D to represent the vector of unit counts. Therefore, we define $D \triangleq \mathbf{x} \in \mathbb{R}^n$, $Q \triangleq \mathbf{W} \in \mathbb{R}^{m\times n}$ ("$\triangleq$" means define). The query batch Q can be answered directly by:

$$Q(D) \triangleq \mathbf{W}\mathbf{x} = \left(\sum_j \mathbf{W}_{1j}\mathbf{x}_j, \ldots, \sum_j \mathbf{W}_{mj}\mathbf{x}_j\right)^T \in \mathbb{R}^{m\times 1}$$

Given a workload matrix $\mathbf{W}$, the worse-case expected squared error of a mechanism $\mathcal{M}$ is defined as [12, 14, 17]:

$$err(\mathcal{M}; \mathbf{W}) \triangleq \max_{\mathbf{x}\in\mathbb{R}^n} \mathbb{E}[\|\mathcal{M}(\mathbf{x}) - \mathbf{W}\mathbf{x}\|_2^2]$$

where the expectation is taken over the randomness of $\mathcal{M}$. Without information of the underlying dataset, the lowest error achievable by any differentially private mechanism for the query matrix $\mathbf{W}$ and database is:

$$opt(\mathbf{W}) = \min_{\mathcal{M}} \; err(\mathcal{M}; \mathbf{W}) \qquad (1)$$

where the infimum is taken over all differentially private mechanisms. If a mechanism $\mathcal{M}$ minimizes the objective value in Eq (1), it is the optimal linear counting query processing mechanism, in the sense that without any prior information of the sensitive data, it achieves the lowest expected error.

Matrix Mechanism. The first solution for answering batch linear aggregate queries under differential privacy is the matrix mechanism [12]. The main idea is that instead of answering the workload queries $\mathbf{W}$ directly, the mechanism first answers a different set of r queries under differential privacy, and then combine their results to answer $\mathbf{W}$. Let matrix $\mathbf{A}$ represent the strategy queries, where each row represent a query and each column represent a unit count. Then, according to the Gaussian mechanism, $\mathbf{A}$ can be answered using $\mathbf{Ax} + \tilde{\mathbf{b}}$ under (ϵ, δ)-differentially privacy, where $\tilde{\mathbf{b}}$ denotes an m dimensional Gaussian random variable with scale $||\mathbf{A}||_{2,\infty}\sqrt{2\ln(2/\delta)}/\epsilon$, and $\|\mathbf{A}\|_{p,\infty}$ is the maximum ℓ_p norm among all column vectors of $\mathbf{A}$. Accordingly, the matrix mechanism answers $\mathbf{W}$ by:

$$\mathcal{M}(\mathbf{x}) = \mathbf{W}(\mathbf{x} + \mathbf{A}^{\dagger}\tilde{\mathbf{b}}) \tag{2}$$

where $\mathbf{A}^{\dagger}$ is the Moore-Penrose pseudo-inverse of $\mathbf{A}$.

Based on Eq (2), Li et al. [12] formalize the strategy searching problem for batch linear counting query processing in Eq(1) into the following nonlinear optimization problem:

$$\min_{\mathbf{A}\setminus\{\mathbf{0}\}} J(\mathbf{A}) \triangleq \|\mathbf{A}\|_{p,\infty}^2 \text{tr}(\mathbf{W}\mathbf{A}^{\dagger}\mathbf{A}^{\dagger T}\mathbf{W}^T). \tag{3}$$

In the above optimization program, p can be either 1 or 2, and the method in [12] applies to both exact and approximate differential privacy. This optimization program, however is rather difficult to solve. The pseudoinverse of $\mathbf{A}^{\dagger}$ of $\mathbf{A}$ involved in Program (3) is not a continuous function, as it jumps around when $\mathbf{A}$ is ill-conditioned. Therefore, $\mathbf{A}^{\dagger}$ does not have a derivative, and we cannot solve the problem with simple gradient descent. As pointed out in [24], the solutions in [12] are either prohibitively expensive (which needs to iteratively solve a pair of related semidefinite programs that incurs $\mathcal{O}(m^3n^3)$ computational costs), or ineffective (which rarely obtains strategies that outperform naive methods).

Low-Rank Mechanism. Yuan et al. [24] propose the Low-Rank Mechanism (LRM), which formulates the batch query problem as the following low-rank matrix factorization problem:

$$\min_{\mathbf{B},\mathbf{L}} \text{tr}(\mathbf{B}^T\mathbf{B})\ s.t.\ \mathbf{W} = \mathbf{BL},\ \|\mathbf{L}\|_{p,\infty} \leq 1 \tag{4}$$

where $\mathbf{B} \in \mathbb{R}^{m\times r}, \mathbf{L} \in \mathbb{R}^{r\times n}$. It can be shown that Program (4) and Program (3) are equivalent to each other; hence, LRM can be viewed as a way to solve the Matrix Mechanism optimization program (to our knowledge, LRM is also the first practical solution for this program). The LRM formulation avoids the pseudo-inverse of the strategy matrix A; however, it is still a non-linear, non-convex constrained optimization program. Hence, it is also difficult to solve. The solution in LRM is a sophisticated numeric method based first-order augmented Lagrangian multipliers (ALM). This solution, however, cannot guarantee to find the globally optimal strategy matrix A, due to the non-convex nature of the problem formulation.

Further, the LRM solution may not converge at all. Specifically, it iteratively updates $\mathbf{B}$ using the formula: $\mathbf{B} \Leftarrow (\beta\mathbf{W}\mathbf{L}^T + \boldsymbol{\pi}\mathbf{L}^T)(\beta\mathbf{L}\mathbf{L}^T + \mathbf{I})^{-1}$, where β is the penalty parameter. When $\mathbf{L}$ is low-rank, according to the rank inequality for matrix multiplication, it leads to: $rank(\mathbf{B}) \leq rank(\mathbf{L})$. Therefore, the equality constraint $\mathbf{W} = \mathbf{BL}$ may never hold since we can never express a full-rank matrix $\mathbf{W}$ with the product of two low-rank ones. When this happens, LRM never converges. For this reason, the initial value of $\mathbf{L}$ needs to be chosen carefully so that it is not low-rank. However, this problem cannot be completed avoided since during the iterations of LRM, the rank of $\mathbf{L}$ may drop. Finally, even in cases where LRM does converge, its convergence rate can be slow, leading to high computational costs as we show in the experiments. In particular, the LRM solution is not necessarily a monotone descent algorithm, meaning that the accuracy of its solutions can fluctuate during the iterations.

Adaptive Mechanism. In order to alleviate the computational overhead of the matrix mechanism, adaptive mechanism (AM) [13] considers the following optimization program:

$$\min_{\boldsymbol{\lambda}\in\mathbb{R}^n} \sum_{i=1}^{n} \frac{\mathbf{d}_i^2}{\boldsymbol{\lambda}_i^2}, s.t.\ (\mathbf{Q}\odot\mathbf{Q})(\boldsymbol{\lambda}\odot\boldsymbol{\lambda}) \leq \mathbf{1} \tag{5}$$

where $\mathbf{Q} \in \mathbb{R}^{m\times n}$ is from the singular value decomposition of the workload matrix $\mathbf{W} = \mathbf{QDP}$ with $\mathbf{D} \in \mathbb{R}^{n\times n}, \mathbf{P} \in \mathbb{R}^{n\times n}$, and $\mathbf{d} = \text{diag}(\mathbf{D}) \in \mathbb{R}^n$, i.e., the diagonal values of $\mathbf{D}$. AM then computes the strategy matrix $\mathbf{A}$ by $\mathbf{A} = \mathbf{Q}\text{diag}(\boldsymbol{\lambda}) \in \mathbb{R}^{m\times n}$, where $\text{diag}(\boldsymbol{\lambda})$ is a diagonal matrix with $\boldsymbol{\lambda}$ as its diagonal values.

The main drawback of AM is that it searches over a reduced subspace of $\mathbf{A}$, since it is limited to a weighted nonnegative combination of the fixed eigen-queries $\mathbf{Q}$. Hence, the candidate strategy matrix $\mathbf{A}$ solved from the optimization problem in (5) is not guaranteed to be the optimal strategy. In fact it is often suboptimal, as shown in the experiments.

Exponential Smoothing Mechanism. Based on a reformulation of matrix mechanism, the Exponential Smoothing Mechanism (ESM) [23] considers solving the following optimization program:

$$\min_{\mathbf{X}\in\mathbb{R}^{n\times n}} \max(\text{diag}(\mathbf{X})) \cdot \text{tr}(\mathbf{W}\mathbf{X}^{-1}\mathbf{W}^T)\ \ s.t.\ \mathbf{X} \succ 0 \tag{6}$$

where $\max$ is a function that retrieves the largest element in a vector. This function is hard to compute since it is non-smooth. The authors use the soft max function $\text{smax}(\mathbf{v}) = \mu \log \sum_i^n(\exp(\frac{\mathbf{v}_i}{\mu}))$ to smooth this term and employ the non-monotone spectral projected gradient descent for optimizing the non-convex but smooth objective function on a positive definiteness constraint set.

One major problem with this method is that Program (6) involves matrix inverse operator, which may cause numerical instability when the final solution (i.e., the strategy matrix) is of low rank. Further, since the problem is not convex, the ESM solution does not guarantee to converge to the global optimum, either.

The proposed solution, presented next, avoids all the drawbacks of previous solutions: it is fast, stable, numerically robust, and most importantly, it guarantees to find the optimal solution.

3. A CONVEX PROBLEM FORMULATION

This section presents the a convex optimization formulation for finding the best strategy for a given workload of linear aggregate queries. The main idea is that instead of solving for the strategy matrix $\mathbf{A}$ that minimizes result error directly, we first solve the optimal value for $\mathbf{X} = \mathbf{A}\mathbf{A}^T$, and then obtain $\mathbf{A}$ accordingly. Note that there can be multiple strategy matrices $\mathbf{A}$ from a given $\mathbf{X} = \mathbf{A}\mathbf{A}^T$, in which case we simply output an arbitrary one, since they all lead to the same overall accuracy for the original workload $\mathbf{W}$. As we show soon, the objective function with respect to $\mathbf{X}$ is convex; hence, the proposed solution is guaranteed to find the global optimum. The re-formulation of the optimization program involves a non-trivial semi-definite programming lifting technique to remove the quadratic term, presented below.

First of all, based on the non-convex model in Program (3), we have the following lemma[1].

LEMMA 1. *Given an arbitrary strategy matrix* $\mathbf{A}$*, we can always construct another strategy* $\mathbf{A}'$ *satisfying (i)* $\|\mathbf{A}'\|_{p,\infty} = 1$ *and (ii)* $J(\mathbf{A}) = J(\mathbf{A}')$*, where* $J(\mathbf{A})$ *is defined in in Program (3).*

By Lemma 1, the following optimization program is equivalent to Program (3).

$$\min_{\mathbf{A}} \langle \mathbf{A}^{\dagger}\mathbf{A}^{\dagger T}, \mathbf{W}^T\mathbf{W}\rangle,\ s.t.\ \|\mathbf{A}\|_{p,\infty} = 1 \tag{7}$$

[1]All proofs can be found in the full version of the paper [22].

This paper focuses on approximate differential privacy where $p = 2$. Moreover, we assume that $\mathbf{V} = \mathbf{W}^T\mathbf{W}$ is full rank. If this assumption does not hold, we simply transform $\mathbf{V}$ into a full rank matrix by adding an identity matrix scaled by θ, where θ approaches zero. Formally, we have:

$$\mathbf{V} = \mathbf{W}^T\mathbf{W} + \theta\mathbf{I} \succ 0 \tag{8}$$

Let $\mathbf{X} = \mathbf{A}^T\mathbf{A} \succ 0$. Using the fact that $(\|\mathbf{A}\|_{2,\infty})^2 = \|\text{diag}(\mathbf{X})\|_\infty$ and $\mathbf{A}^\dagger\mathbf{A}^{\dagger T} = \mathbf{X}^{-1}$, we have the following matrix inverse optimization program (note that $\mathbf{X}$ and $\mathbf{V}$ are both full-rank):

$$\min_{\mathbf{X}} \; F(\mathbf{X}) = \langle \mathbf{X}^{-1}, \mathbf{V}\rangle, \; s.t. \; \text{diag}(\mathbf{X}) \leq \mathbf{1}, \; \mathbf{X} \succ 0. \tag{9}$$

Interestingly, using the fact that $||\mathbf{X}/n|| \leq tr(\mathbf{X}/n) \leq 1$, one can approximate the matrix inverse via Neumann Series [2] and rewrite the objective function in terms of matrix polynomials [3]. Although other convex semi-definite programming reformulations/relaxations exist (discussed in the full version of the paper [22]), we focus on Program (9) and provide convex analysis below.

Convexity of Program (9). Observe that the objective function of Program (9) is not always convex unless some conditions are imposed on $\mathbf{V}$ and $\mathbf{X}$. For instance, in the the one-dimensional case, it reduces to the inversely proportional function $f(x) = \frac{k}{x}$, with $k > 0$. Clearly, $f(x)$ is convex on the strictly positive space and concave on the strictly negative space.

The following lemma states the convexity of Program (9) under appropriate conditions.

LEMMA 2. *Assume that $\mathbf{X} \succ 0$. The function $F(\mathbf{X}) = \langle \mathbf{X}^{-1}, \mathbf{V}\rangle$ is convex (resp., strictly convex) if $\mathbf{V} \succeq 0$ (resp., $\mathbf{V} \succ 0$).*

Since $\mathbf{V}$ is the covariance matrix of $\mathbf{W}$, $\mathbf{V}$ is always positive semidefinite. Therefore, according to the above lemma, the objective function of Program (9) is convex. Furthermore, since $\mathbf{V}$ is strictly positive definite, the objective function $F(\mathbf{X})$ is actually strictly convex. Therefore, there exists a unique optimal solution for Program (9).

Dual program of Program (9). The following lemma describes the dual program of Program (9).

LEMMA 3. *The dual program of Program (9) is the following:*

$$\max_{\mathbf{X},\mathbf{y}} \; -\langle \mathbf{y}, \mathbf{1}\rangle, \; s.t. \; \mathbf{X}diag(\mathbf{y})\mathbf{X} - \mathbf{V} \succeq 0, \; \mathbf{X} \succ 0, \; \mathbf{y} \geq 0.$$

where $\mathbf{y} \in \mathbb{R}^n$ is associated with the inequality constraint $diag(\mathbf{X}) \leq \mathbf{1}$.

Lower and upper bounds for Program (9). Next we establish a lower bound and an upper bound on the objective function of Program (9) for any feasible solution.

LEMMA 4. *For any feasible solution $\mathbf{X}$ in Program (9), its objective value is sandwiched as*

$$\max(2\|\mathbf{W}\|_* - n, \; \|\mathbf{W}\|_*^2/n) + \theta \leq F(\mathbf{X}) \leq \rho^2(\|\mathbf{W}\|_F^2 + \theta n)$$

where $\rho = \max_i \|\mathbf{S}(:, i)\|_2$, $i \in [n]$, and $\mathbf{S}$ comes from the SVD decomposition that $\mathbf{W} = \mathbf{U\Sigma S}$.

The parameter $\theta \geq 0$ serves as regularization of the convex problem. When $\theta > 0$, we always have $\mathbf{V} \succ 0$. As can be seen in our subsequent analysis, the assumption that $\mathbf{V}$ is strictly positive definite is necessary in our algorithm design.

Problem formulation with equality constraints. We next reformulate Program (9) in the following lemma.

[2] $\mathbf{X}^{-1} = \sum_{k=0}^{\infty}(\mathbf{I} - \mathbf{X})^k$, $\forall \; \|\mathbf{X}\| \leq 1$

[3] $F(\mathbf{X}) = \langle (\mathbf{X}/n)^{-1}, \mathbf{V}/n\rangle = \langle \sum_{k=0}^{\infty}(\mathbf{I} - \mathbf{X}/n)^k, \mathbf{V}/n\rangle$

LEMMA 5. *Assume $\mathbf{V} \succ 0$. The optimization problem in Program (9) is equivalent to the following optimization program:*

$$\min_{\mathbf{X}} \; F(\mathbf{X}) = \langle \mathbf{X}^{-1}, \mathbf{V}\rangle, \; s.t. \; diag(\mathbf{X}) = \mathbf{1}, \; \mathbf{X} \succ 0. \tag{10}$$

Program (10) is much more attractive than Program (9) since the equality constraint is easier to handle than the inequality constraint. As can be seen in our algorithm design below, this equality constraint can be explicitly enforced with suitable initialization. Hence, in the rest of the paper, we focus on solving Program (10).

First-order and second-order analysis. It is not hard to verify that the first-order and second-order derivatives of the objective function $F(\mathbf{X})$ can be expressed as (see page 700 in [1]):

$$\begin{aligned} G(\mathbf{X}) &= -\mathbf{X}^{-1}\mathbf{V}\mathbf{X}^{-1}, \\ H(\mathbf{X}) &= -G(\mathbf{X}) \otimes \mathbf{X}^{-1} - \mathbf{X}^{-1} \otimes G(\mathbf{X}) \end{aligned} \tag{11}$$

Since our method (described soon) is a greedy descent algorithm, we restrict our discussions on the level set $\mathcal{X}$ which is defined as:

$$\mathcal{X} \triangleq \{\mathbf{X} | F(\mathbf{X}) \leq F(\mathbf{X}^0), \text{ and } \text{diag}(\mathbf{X}) = \mathbf{1}, \text{ and } \mathbf{X} \succ 0\}$$

We now analyze bounds for the eigenvalues of the solution in Program (10), as well as bounds for the eigenvalues of the Hessian matrix and the gradient matrix of the objective function in Program (10). The following lemma shows that the eigenvalues of the solution in Program (10) are bounded.

LEMMA 6. *For any $\mathbf{X} \in \mathcal{X}$, there exist some strictly positive constants C_1 and C_2 such that $C_1\mathbf{I} \preceq \mathbf{X} \preceq C_2\mathbf{I}$ where $C_1 = (\frac{F(\mathbf{X}^0)}{\boldsymbol{\lambda}_1(\mathbf{V})} - 1 + \frac{1}{n})^{-1}$ and $C_2 = n$.*

The next lemma shows the the eigenvalues of the Hessian matrix and the gradient matrix of the objective function in Program (10) are also bounded.

LEMMA 7. *For any $\mathbf{X} \in \mathcal{X}$, there exist some strictly positive constants C_3, C_4, C_5 and C_6 such that $C_3\mathbf{I} \preceq H(\mathbf{X}) \preceq C_4\mathbf{I}$ and $C_5\mathbf{I} \preceq G(\mathbf{X}) \preceq C_6\mathbf{I}$, where $C_3 = \frac{\boldsymbol{\lambda}_1(\mathbf{V})}{C_2^3(\mathbf{X})}$, $C_4 = \frac{\boldsymbol{\lambda}_n(\mathbf{V})}{C_1^3(\mathbf{X})}$, $C_5 = \frac{\boldsymbol{\lambda}_1(\mathbf{V})}{C_2^2(\mathbf{X})}$, $C_6 = \frac{\boldsymbol{\lambda}_n(\mathbf{V})}{C_1^2(\mathbf{X})}$.*

A self-concordant function [16] is a function $f : \mathbb{R} \to \mathbb{R}$ for which $|f'''(x)| \leq 2f''(x)^{3/2}$ in the affective domain. It is useful in the analysis of Newton's method. A self-concordant barrier function is used to develop interior point methods for convex optimization.

Self-Concordance Property. The following lemma establishes the self-concordance property of Program (10).

LEMMA 8. *The objective function $\tilde{F}(\mathbf{X}) = \frac{C^2}{4}F(\mathbf{X}) = \frac{C^2}{4} \cdot \langle \mathbf{X}^{-1}, \mathbf{V}\rangle$ with $\mathbf{X} \in \mathcal{X}$ is a standard self-concordant function, where C is a strictly positive constant with $C \triangleq \frac{6C_2^3 tr(\mathbf{V})^{-1/2}}{2^{3/2}C_1^3}$.*

The self-concordance plays a crucial role in our algorithm design and convergence analysis. First, self-concordance ensures that the current solution is always in the interior of the constraint set $\mathbf{X} \succ 0$ [16] , which makes it possible for us to design a new Cholesky decomposition-based algorithm that can avoid eigenvalue decomposition[4]. Second, self-concordance controls the rate at which the second derivative of a function changes, and it provides a checkable sufficient condition to ensure that our method converges to the global solution with (local) quadratic convergence rate.

[4] Although Cholesky decomposition and eigenvalue decomposition share the same computational complexity ($\mathcal{O}(n^3)$) for factorizing a positive definite matrix of size n, in practice Cholesky decomposition is often significantly faster than eigenvalue decomposition (e.g. by about 50 times for a square matrix of size $n = 5000$).

Algorithm 1 **Algorithm COA for Solving Program (10)**

```
 1: Input: $\theta > 0$ and $\mathbf{X}^0$ such that $\mathbf{X}^0 \succ 0, \text{diag}(\mathbf{X}^0) = \mathbf{1}$
 2: Output: $\mathbf{X}^k$
 3: Initialize $k = 0$
 4: while not converge do
 5:     Solve the following subproblem by Algorithm 2:
            $$\mathbf{D}^k \Leftarrow \arg\min_{\mathbf{\Delta}} f(\mathbf{\Delta}; \mathbf{X}^k), \; s.t. \; \text{diag}(\mathbf{X}^k + \mathbf{\Delta}) = \mathbf{1} \quad (12)$$
 6:     Perform step-size search to get $\alpha^k$ such that:
 7:       (1) $\mathbf{X}^{k+1} = \mathbf{X}^k + \alpha^k \mathbf{D}^k$ is positive definite and
 8:       (2) there is sufficient decrease in the objective.
 9:     if $\mathbf{X}^k$ is an optimal solution of Program (10) then
10:         terminate and output $\mathbf{X}^k$
11:     Increment $k$ by 1
```

Algorithm 2 **A Modified Conjugate Gradient for Finding $\mathbf{D}$ as in Program (15)**

```
 1: Input: $\mathbf{Z} = (\mathbf{X}^k)^{-1}$, and current gradient $\mathbf{G} = G(\mathbf{X}^k)$, Specify the maximum iteration $T \in \mathbb{N}$
 2: Output: Newton direction $\mathbf{D} \in \mathbb{R}^{n \times n}$
 3: $\mathbf{D} = 0, \mathbf{R} = -\mathbf{G} + \mathbf{ZDG} + \mathbf{GDZ}$
 4: Set $\mathbf{D}_{ij} = 0, \; \mathbf{R}_{ij} = 0, \; \forall i = j, i, j \in [n]$
 5: $\mathbf{P} = \mathbf{R}, r_{old} = \langle \mathbf{R}, \mathbf{R} \rangle$
 6: for $l = 0$ to $T$ do
 7:     $\mathbf{B} = -\mathbf{G} + \mathbf{ZDG} + \mathbf{GDZ}, \; \alpha = \frac{r_{old}}{\langle \mathbf{P}, \mathbf{B} \rangle}$
 8:     $\mathbf{D} = \mathbf{D} + \alpha \mathbf{P}, \mathbf{R} = \mathbf{R} - \alpha \mathbf{B}$
 9:     Set $\mathbf{D}_{ij} = 0, \; \mathbf{R}_{ij} = 0, \; \forall i = j, \; i, j \in [n]$
10:     $r_{new} = \langle \mathbf{R}, \mathbf{R} \rangle, \; \mathbf{P} = \mathbf{R} + \frac{r_{new}}{r_{old}} \mathbf{P}, r_{old} = r_{new}$
11: return $\mathbf{D}$
```

4. OPTIMIZATION ALGORITHM

In this section, we provide a Newton-like algorithm COA to solve Program (10). We first show how to find the search direction and the step size in Sections 4.1 and 4.2, respectively. Then we study the convergence property of COA in Section 4.3. Finally, we present a homotopy algorithm to further accelerate the convergence. For notational convenience, we use the shorthand notation $F^k = F(\mathbf{X}^k)$, $\mathbf{G}^k = G(\mathbf{X}^k)$, $\mathbf{H}^k = H(\mathbf{X}^k)$, and $\mathbf{D} = D(\mathbf{X}^k)$ to denote the objective value, first-order gradient, hessian matrix and the search direction at the point $\mathbf{X}^k$, respectively.

Following the approach of [20, 9, 25], we build a quadratic approximation around any solution $\mathbf{X}^k$ for the objective function $F(\mathbf{X})$ by considering its second-order Taylor expansion:

$$f(\mathbf{\Delta}; \mathbf{X}^k) = F^k + \langle \mathbf{\Delta}, \mathbf{G}^k \rangle + \frac{1}{2} \text{vec}(\mathbf{\Delta})^T \mathbf{H}^k \text{vec}(\mathbf{\Delta}). \quad (13)$$

Therefore, the Newton direction $\mathbf{D}^k$ for the smooth objective functon $F(\mathbf{X})$ can then be written as the solution of the following equality constrained quadratic program:

$$\mathbf{D}^k = \arg\min_{\mathbf{\Delta}} f(\mathbf{\Delta}; \mathbf{X}^k), \; s.t. \; \text{diag}(\mathbf{X}^k + \mathbf{\Delta}) = \mathbf{1}, \quad (14)$$

After the direction $\mathbf{D}^k$ is computed, we employ an Arimijo-rule based step size selection to ensure positive definiteness and sufficient descent of the next iterate. We summarize our algorithm COA in Algorithm 1. Note that the initial point $\mathbf{X}^0$ has to be a feasible solution, thus $\mathbf{X}^0 \succ 0$ and $\text{diag}(\mathbf{X}^0) = \mathbf{1}$. Moreover, the positive definiteness of all the following iterates $\mathbf{X}^k$ will be guaranteed by the step size selection procedure (refer to step 7 in Algorithm 1).

4.1 Computing the Search Direction

This subsection is devoted to finding the search direction in Eq (14). With the choice of $\mathbf{X}^0 \succ 0$ and $\text{diag}(\mathbf{X}^0) = \mathbf{1}$, Eq(14) reduces to the following optimization program:

$$\min_{\mathbf{\Delta}} \; \langle \mathbf{\Delta}, \mathbf{G}^k \rangle + \frac{1}{2} \text{vec}(\mathbf{\Delta})^T \mathbf{H}^k \text{vec}(\mathbf{\Delta}), \; s.t. \; \text{diag}(\mathbf{\Delta}) = \mathbf{0} \quad (15)$$

At first glance, Program (15) is challenging. First, this is a constrained optimization program with $n \times n$ variables and n equality constraints. Second, the optimization problem involves computing and storing an $n^2 \times n^2$ Hessian matrix $\mathbf{H}^k$, which is a daunting task in algorithm design.

We carefully analyze Problem (15) and propose the following solutions. For the first issue, Eq (15) is actually a unconstrained quadratic program with $n \times (n-1)$ variable. In order to handle the diagonal variables of $\mathbf{\Delta}$, one can explicitly enforce the diagonal entries of current solution and its gradient to $\mathbf{0}$. Therefore, the constraint $\text{diag}(\mathbf{\Delta}) = \mathbf{0}$ can always be guaranteed. This implies that linear conjugate gradient method can be used to solve Problem (15). For the second issue, we can make good use of the Kronecker product structure of the Hessian matrix. We note that $(\mathbf{A} \otimes \mathbf{B}) \, vec(\mathbf{C}) = vec(\mathbf{BCA}), \forall \mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{n \times n}$. Given a vector $vec(\mathbf{D}) \in \mathbb{R}^{n^2 \times 1}$, using the fact that the Hessian matrix can be computed as $\mathbf{H} = -\mathbf{G} \otimes \mathbf{X}^{-1} - \mathbf{X}^{-1} \otimes \mathbf{G}$, the Hessian-vector product can be computed efficiently as: $\mathbf{H} vec(\mathbf{D}) = vec(-\mathbf{GDX}^{-1} - \mathbf{X}^{-1}\mathbf{DG})$, which only involves matrix-matrix computation. Our modified linear conjugate gradient method for finding the search direction is summarized in Algorithm 2. Note that the algorithm involves a parameter T controlling the maximum number of iterations. The specific value of T does not affect the correctness of COA, only its efficiency. Through experiments we found that a value of $T = 5$ usually leads to good overall efficiency of COA.

4.2 Computing the Step Size

After the Newton direction $\mathbf{D}$ is found, we need to compute a step size $\alpha \in (0, 1]$ that ensures positive definiteness of the next iterate $\mathbf{X} + \alpha \mathbf{D}$ and leads to a sufficient decrease of the objective function. We use Armijo's rule and try step size $\alpha \in \{\beta^0, \beta^1, ...\}$ with a constant decrease rate $0 < \beta < 1$ until we find the smallest $t \in \mathbb{N}$ with $\alpha = \beta^t$ such that $\mathbf{X} + \alpha \mathbf{D}$ is (i) positive definite, and (ii) satisfies the following sufficient decrease condition [20]:

$$F(\mathbf{X}^k + \alpha^k \mathbf{D}^k) \leq F(\mathbf{X}^k) + \alpha^k \sigma \langle \mathbf{G}^k, \mathbf{D}^k \rangle, \quad (16)$$

where $0 < \sigma < 0.5$. We choose $\beta = 0.1$ and $\sigma = 0.25$ in our experiments.

We verify positive definiteness of the solution while computing its Cholesky factorization (takes $\frac{1}{3}n^3$ flops). We remark that the Cholesky factorization dominates the computational cost in the step-size computations. To reduce the computation cost, we can reuse the Cholesky factor in the previous iteration when evaluating the objective function (that requires the computation of $\mathbf{X}^{-1}$). The decrease condition in Eq (16) has been considered in [20] to ensure that the objective value not only decreases but also decreases by a certain amount $\alpha^k \sigma \langle \mathbf{G}^k, \mathbf{D}^k \rangle$, where $\langle \mathbf{G}^k, \mathbf{D}^k \rangle$ measures the optimality of the current solution.

The following lemma provides some theoretical insights of the line search program. It states that a strictly positive step size can always be achieved in Algorithm 1. This property is crucial in our global convergence analysis of the algorithm.

LEMMA 9. *There exists a strictly positive constant $\alpha < \min(1, \frac{C_1}{C_7}, C_8)$ such that the positive definiteness and sufficient descent conditions (in step 7-8 of Algorithm 1) are satisfied. Here $C_7 \triangleq \frac{2\lambda_n(\mathbf{V})}{C_1^2 C_3}$ and $C_8 \triangleq \frac{2(1-\sigma)C_3}{C_4}$ are some constants which are independent of the current solution $\mathbf{X}^k$.*

The following lemma shows that a full Newton step size will be selected eventually. This is useful for the proof of local quadratic convergence.

LEMMA 10. *If $\mathbf{X}^k$ is close enough to global optimal solution such that $\|\mathbf{D}^k\| \leq \min(\frac{3.24}{C^2 C_4}, \frac{(2\sigma+1)^2}{C^6 C^2})$, the line search condition will be satisfied with step size $\alpha^k = 1$.*

4.3 Theoretical Analysis

First, we provide convergence properties of Algorithm 1. We prove that Algorithm 1 always converges to the global optimum, and then analyze its convergence rate. Our convergence analysis is mainly based on the proximal point gradient method [20, 9] for composite function optimization in the literature. Specifically, we have the following results (proofs appear in the full version [22]):

THEOREM 1. ***Global Convergence of Algorithm 1.*** *Let $\{\mathbf{X}^k\}$ be sequences generated by Algorithm 1. Then $F(\mathbf{X}^k)$ is nonincreasing and converges to the global optimal solution.*

THEOREM 2. ***Global Linear Convergence Rate of Algorithm 1.*** *Let $\{\mathbf{X}^k\}$ be sequences generated by Algorithm 1, Then $\{\mathbf{X}^k\}$ converges linearly to the global optimal solution.*

THEOREM 3. ***Local Quadratic Convergence Rate of Algorithm 1.*** *Let $\{\mathbf{X}^k\}$ be sequences generated by Algorithm 1. When $\mathbf{X}^k$ is sufficiently close to the global optimal solution, then $\{\mathbf{X}^k\}$ converges quadratically to the global optimal solution.*

It is worth mentioning that Algorithm 1 is the *first polynomial algorithm* for linear query processing under approximate differential privacy with a provable global optimum guarantee.

Next we analyze the time complexity of our algorithm. Assume that COA converges within N_{coa} outer iterations (Algorithm 1) and T_{coa} inner iterations (Algorithm 2). Due to the $\mathcal{O}(n^3)$ complexity for Cholesky factorization (where n is the number of unit counts), the total complexity of COA is $\mathcal{O}(N_{\text{coa}} \cdot T_{\text{coa}} \cdot n^3)$. Note that the running time of COA is independent of the number of queries m. In contrast, the current state-of-the-art LRM has time complexity $\mathcal{O}(N_{\text{lrm}} \cdot T_{\text{lrm}} \cdot \min(m,n)^2 \cdot (m+n))$ (where N_{lrm} and T_{lrm} are the number of outer and inner iterations of LRM, respectively), which involves both n and m. Note that $(N_{\text{coa}} \cdot T_{\text{coa}})$ in the big $\mathcal{O}$ notation is often much smaller than $(N_{\text{lrm}} \cdot T_{\text{lrm}})$ in practice, due to the quadratic convergence rate of COA. According to our experiments, typically COA converges with $N_{\text{coa}} \leq 10$ and $T_{\text{coa}} \leq 5$.

4.4 A Homotopy Algorithm

In Algorithm 1, we assume that $\mathbf{V}$ is positive definite. If this is not true, one can consider adding a deceasing regularization parameter to the diagonal entries of $\mathbf{V}$. We present a homotopy algorithm for solving Program (9) with θ approaching 0 in Algorithm 3.

The homotopy algorithm used in [19, 5] have shown the advantages of continuation method in speeding up solving large-scale optimization problems. In continuation method, a sequence of optimization problems with deceasing regularization parameter is solved until a sufficiently small value is arrived. The solution of each optimization is used as the warm start for the next iteration.

In Eq (8), a smaller θ is always preferred because it results in more accurate approximation of the original optimization in Program (9). However, it also implies a slower convergence rate, according to our convergence analysis. Hence the computational cost of our algorithm is high when small θ is selected. In Algorithm 3, a series of problems with decreasing regularization parameter θ are solved by using Algorithm 1, and the solution of each run of Algorithm 1 is used as the initial solution $\mathbf{X}^0$ of the next iteration. In this paper, Algorithm 3 starts from a large $\theta^0 = 1$, and it stops when the preferred $\theta \leq 10^{-10}$ arrives.

Algorithm 3 A Homotopy Algorithm for Solving Eq (9) with θ approaching 0.

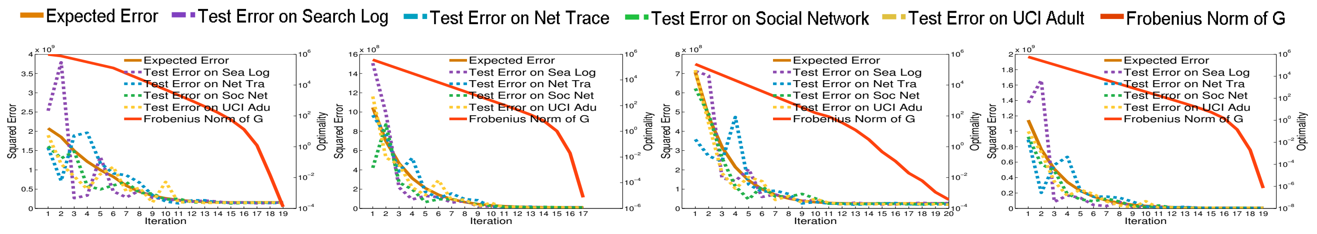
```
1: Input: workload matrix $\mathbf{W}$
2: Output: $\mathbf{X}$
3: Specify the maximum iteration $T = 10$
4: Initialize $\mathbf{X}^0 = \mathbf{I}$, $\theta^0 = 1$
5: for $i = 0$ to $T$ do
6:     Apply Algorithm 1 with $\theta^i$ and $\mathbf{X}^i$ to obtain $\mathbf{X}^{i+1}$
7:     $\theta^{i+1} = \theta^i \times 0.1$
```

5. EXPERIMENTS

This section experimentally evaluates the effectiveness of the proposed convex optimization algorithm COA for linear aggregate processing under approximate differential privacy. We compare COA with six existing methods: Gaussian Mechanism (GM) [15], Wavelet Mechanism (WM) [21], Hierarchical Mechanism (HM) [7], Exponential Smoothing Mechanism (ESM) [23, 12], Adaptive Mechanism (AM) [13, 12] and Low-Rank Mechanism (LRM) [23, 24]. Qardaji et al. [18] proposed an improved version of HM by carefully selecting the branching factor. Similar to HM, this method focuses on range processing, and there is no guarantee on result quality for general linear aggregates. A detailed experimental comparison with [18] is left as future work. Moreover, we also compare with a recent hybrid data- and workload-aware method [11] which is designed only for range queries and exact differential privacy. Since a previous study [24] has shown that LRM significantly outperforms MWEM, we do not compare with Exponential Mechanism with Multiplicative Weights update (MWEM). Although the batch query processing problem under approximate differential privacy in Program (9) can be reformulated as a standard semi-definite programming problem which can be solved by interior point solvers, we do not compare with it either since such method requires prohibitively high CPU time and memory consumption even for one single (Newton) iteration.

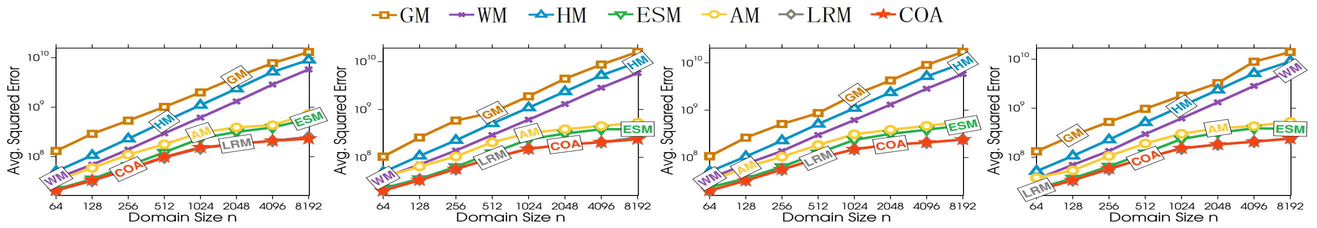
For AM, we employ the Python implementation obtained from the authors' website: http://cs.umass.edu/~chaoli. We use the default stopping criterion provided by the authors. For ESM and LRM, we use the Mablab code provided by the authors: http://yuanganzhao.weebly.com/. For COA, we implement the algorithm in Matlab (refer to the Appendix in [22]) and only report the results of Algorithm 1 with the parameter $\theta = 10^{-3}$. We performed all experiments on a desktop PC with an Intel quad-core 2.50 GHz CPU and 4GBytes RAM. In each experiment, every algorithm is executed 20 times and the average performance is reported.

Following the experimental settings in [24], we use four real-world data sets (*Search Log*, *Net Trace*, *Social Network* and *UCI Adult*) and fours different types of workloads (*WDiscrete*, *WRange*, *WMarginal* and *WRelated*). In *WDiscrete*, each entry is a random variable follows the bernoulli distribution; in *WRange*, each query sums the unit counts in a range whose start and end points are randomly generated following the uniform distribution. *WMarginal* contains queries uniformly sampled from the set of all two-way marginals. For *WRelated*, we generate workload matrix by low-rank matrix multiplication [24]. Moreover, we measure average squared error and computation time of all the methods. Here the average squared error is the average squared ℓ_2 distance between the exact query answers and the noisy answers. The privacy parameters are set to $\epsilon = 0.1$, $\delta = 0.0001$ in our experiments for



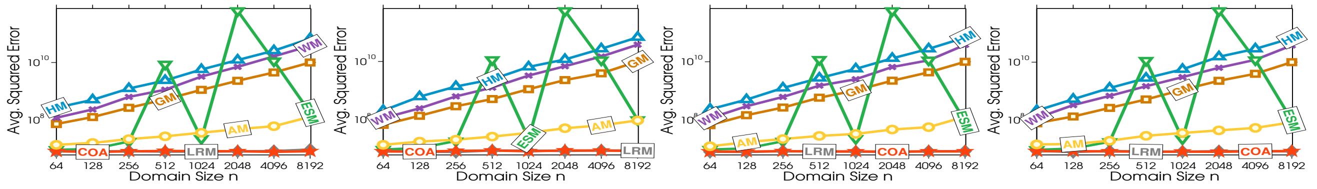
(a) WDiscrete (b) WMarginal (c) WRange (d) WRelated

Figure 1: Convergence behavior of the proposed convex optimization algorithm (Algorithm 1).



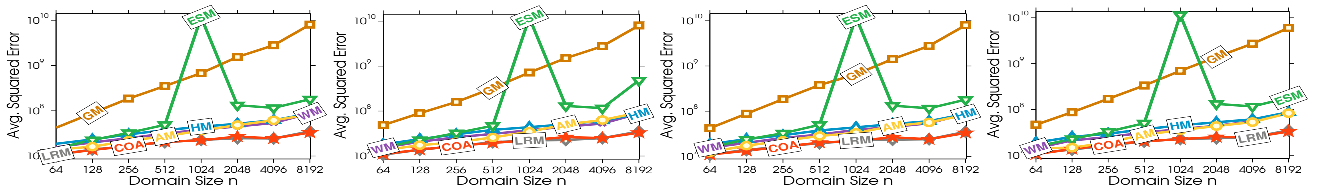
(a) Search Log (b) Net Trace (c) Social Network (d) UCI Adult

Figure 2: Effect of varying domain size n with $m = 1024$ on workload *WDiscrete*.



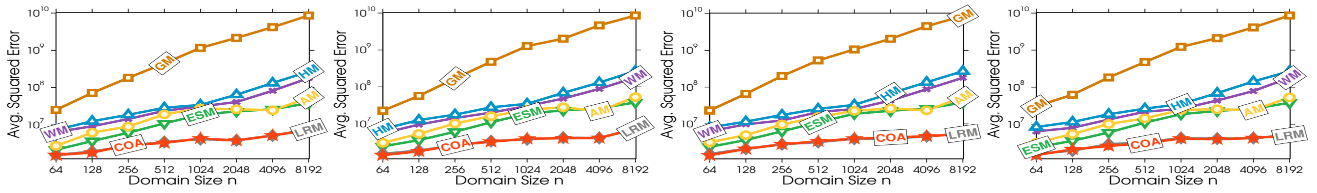
(a) Search Log (b) Net Trace (c) Social Network (d) UCI Adult

Figure 3: Effect of varying domain size n with $m = 1024$ on workload *WMarginal*.



(a) Search Log (b) Net Trace (c) Social Network (d) UCI Adult

Figure 4: Effect of varying domain size n with $m = 1024$ on workload *WRange*.



(a) Search Log (b) Net Trace (c) Social Network (d) UCI Adult

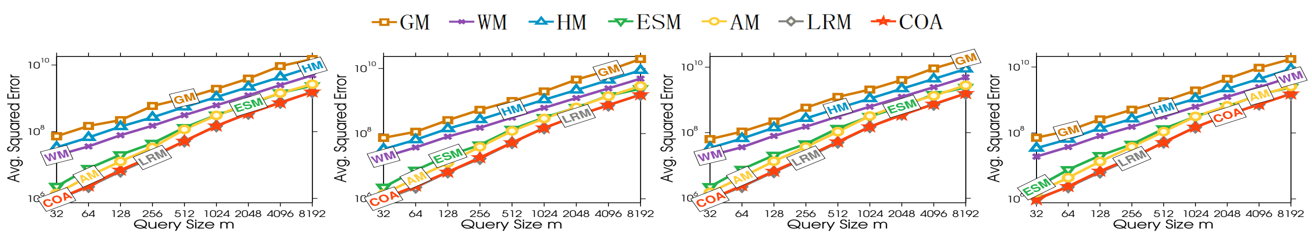
Figure 5: Effect of varying domain size n with $m = 1024$ on workload *WRelated*.

all methods, except for DAWA, which has $\epsilon = 0.1,\ \delta = 0$ since it answers queries under exact differential privacy.

Convergence Behavior of COA: Firstly, we verify the convergence property of COA using all the datasets on all the workloads. We record the objective value (i.e. the expected error), the optimality measure (i.e. $\|\mathbf{G}^k\|_F$) and the test error on four datasets at every iteration k and plot these results in Figure 1.

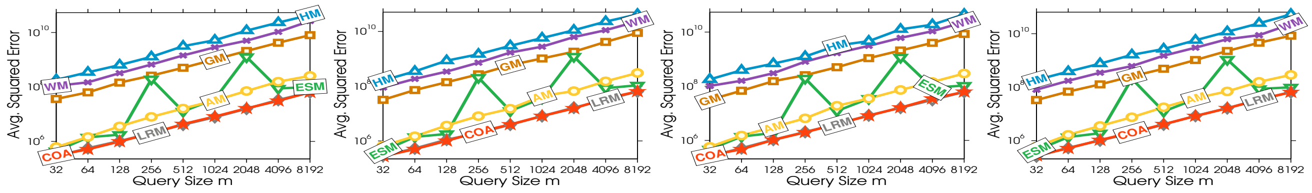
We make three important observations from these results. (i) The objective value and optimality measure decrease monotonically. This is because our method is a greedy descent algorithm. (ii) The test errors do not necessarily decrease monotonically but tend to decrease iteratively. This is because we add random gaussian noise to the results and the average squared error is expected to decrease. (iii) The objective values stabilize after the 10th iteration, which means that our algorithm has converged, and the decrease of the error is negligible after the 10th iteration. This implies that one may use a looser stopping criterion without sacrificing accuracy.

Impact of Varying Number of Unit Counts: We now evaluate the accuracy performance of all mechanisms with varying domain size n from 64 to 8192, after fixing the number of queries m to 1024. We report the results of all mechanisms on the 4 different workloads in Figures 2, 3, 4 and 5, respectively. We have the following observations. (i) COA obtains comparable results with LRM, the current state of the art. Part of the reason may be that, the random initialization strategy makes LRM avoid undesirable local minima. In addition, COA and LRM achieve the best performance in all settings. Their improvement over the naive GM is over two orders of magnitude, especially when the domain size is large. (ii) WM and



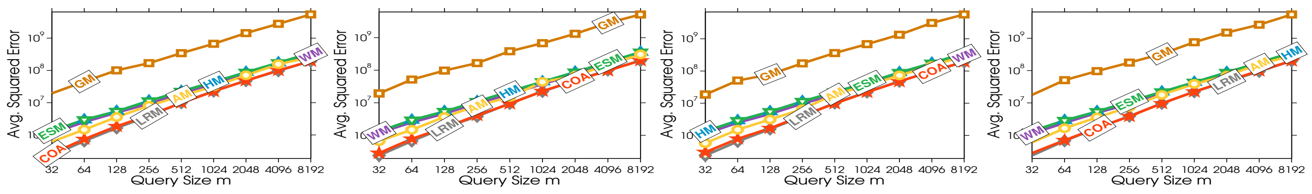
(a) Search Log (b) Net Trace (c) Social Network (d) UCI Adult

Figure 6: Effect of varying number of queries m with $n = 512$ on workload *WDiscrete*.



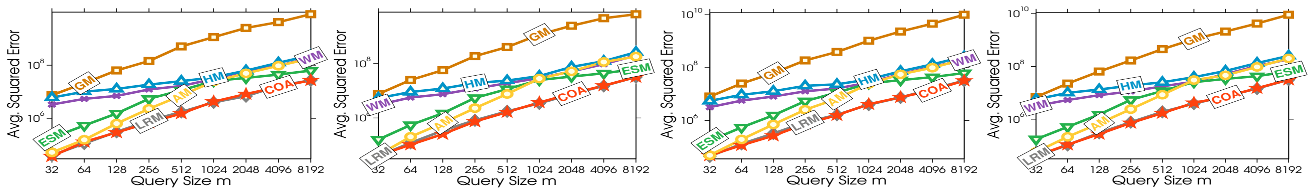
(a) Search Log (b) Net Trace (c) Social Network (d) UCI Adult

Figure 7: Effect of varying number of queries m with $n = 512$ on workload *WMarginal*.



(a) Search Log (b) Net Trace (c) Social Network (d) UCI Adult

Figure 8: Effect of varying number of queries m with $n = 512$ on workload *WRange*.



(a) Search Log (b) Net Trace (c) Social Network (d) UCI Adult

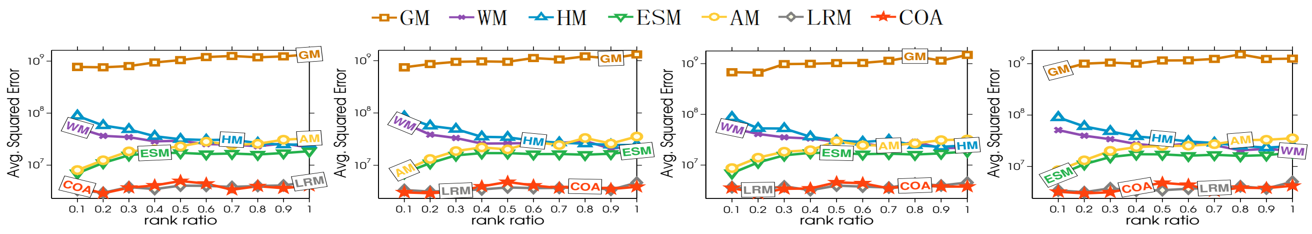
Figure 9: Effect of varying number of queries m with $n = 512$ on workload *WRelated*.

HM obtain similar accuracy on *WRange* and they are comparable to COA and LRM. This is because they are designed for range queries optimization. (iii) AM and ESM have similar accuracy and they are usually strictly worse than COA and LRM. Moreover, the accuracy of AM and ESM is rather unstable on workload *WMarginal*. For ESM, this instability is caused by numerical errors in the matrix inverse operation, which can be high when the final solution matrix is low-rank. Finally, AM searches in a reduced subspace for the optimal strategy matrix, leading to suboptimal solutions with unstable quality.

Impact of Varying Number of Queries: In part of the section, we test the impact of varying the query set cardinality m from 32 to 8192 with n fixed to 512. The accuracy results of all mechanisms on the 4 different workloads are reported in Figures 6, 7, 8 and 9. We have the following observations. (i) COA and LR-M have similar performance and they consistently outperform all the other methods in all test cases. (ii) On *WDiscrete* and *WRange* workloads, AM and ESM show comparable performance, which is much worse performance than COA and LRM. (iii) On *WDiscrete*, *WRange* and *WRelated* workload, WM and HM improve upon the naive Gaussian mechanism; however, on *WMarginal*, WM and H-M incur higher errors than GM. AM and ESM again exhibit similar performance, which is often better than that of WM, HM, and GM.

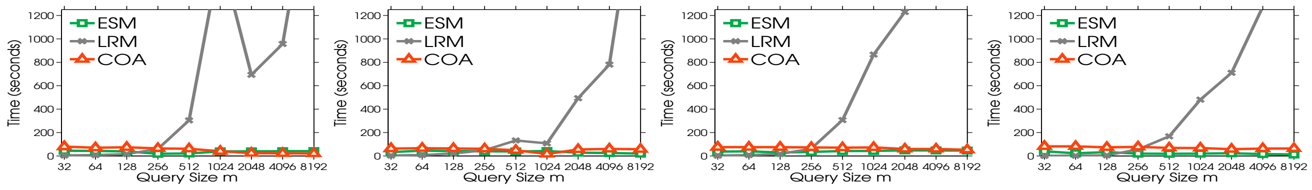
Impact of Varying Rank of Workload: Past studies [23, 24] show that it is possible to reduce the expected error when the workload matrix has low rank. In this set of experiments, we manually control the rank of workload W to verify this claim. Recall that the parameter s determines the size of the matrix $\mathbf{C} \in \mathbb{R}^{m\times s}$ and the size of the matrix $\mathbf{A} \in \mathbb{R}^{s\times n}$ during the generation of the *WRelated* workload. When $\mathbf{C}$ and $\mathbf{A}$ contain only independent rows/columns, s is exactly the rank of the workload matrix $\mathbf{W} = \mathbf{CA}$. In Figure 10, we vary s from $0.1 \times \min(m, n)$ to $1 \times \min(m, n)$. We observe that both LRM and COA outperform all other methods by at least one order of magnitude. With increasing s, the performance gap gradually closes. Meanwhile, COA's performance is again comparable to LRM.

Running Time Evaluations: We now demonstrate the efficiency of LRM, ESM and COA for the 4 different types of workloads. Other methods, such as WM and HM, requires negligible time since they are essentially heuristics without complex optimization computations. From our experiments we obtain the following results. (i) In Figure 11, we vary m from 32 to 8192 and fix n to 1024. COA requires the same running time regardless of the number of queries m, whereas the efficiency of LRM deteriorates with increasing m. (ii) In Figure 12, we vary n from 32 to 8192 and fix m to 1024. We observe that COA is more efficient than LRM when n is relatively small (i.e., $n < 5000$). This is mainly because COA converges with much fewer iterations than LRM. Specifically, we found through manual inspection that COA converges within about $N_{\text{coa}} = 10$ outer iterations (refer to Figure 1) and $T_{\text{coa}} = 5$ inner



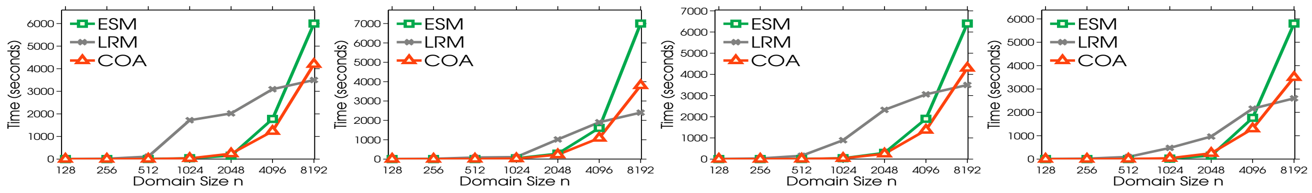
(a) Net Trace (b) Search Log (c) Social Network (d) UCI Adult

Figure 10: Effect of varying s and fixed $m = 1024$, $n = 1024$ on different datasets.



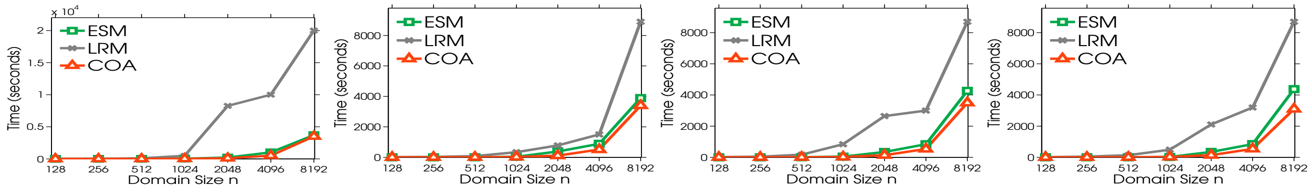
(a) WDiscrete (b) WMarginal (c) WRange (d) WRelated

Figure 11: Running time comparisons with varying m and fixed $n = 1024$ for different workloads.



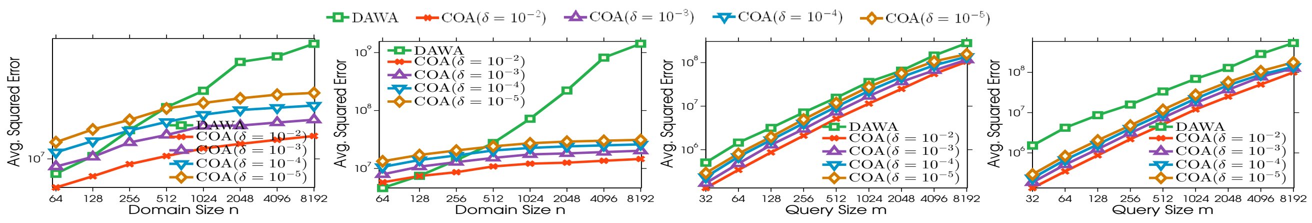
(a) WDiscrete (b) WMarginal (c) WRange (d) WRelated

Figure 12: Running time comparisons with varying n and fixed $m = 1024$ for different workloads.



(a) WDiscrete (b) WMarginal (c) WRange (d) WRelated

Figure 13: Running time comparisons with varying n and fixed $m = 2048$ for different workloads.



(a) Social Network (b) Random Uniform (c) Social Network (d) Random Uniform

Figure 14: (a,b) Effect of varying domain size n with $m = 1024$ on workload WRange. (c,d) Effect of varying number of queries m with n=1024 on workload WRange.

iterations (refer to our Matlab code in the full version [22]). In contract, LRM often takes about $N_{\text{lrm}} = 200$ outer iterations and about $T_{\text{lrm}} = 50$ inner iterations to converge. When n is very large (e.g., when $n = 8192$) and m is relatively small (1024), COA may run slower than LRM due to the former's cubic runtime complexity with respect to the domain size n. (iii) In Figure 13, we vary n from 32 to 8192 and fix m to a lager value 2048. We observe that COA is much more efficient than LRM for all values of n. This is because the runtime of COA is independent of m while LRM scale quadratically with $\min(m, n)$, and COA has quadratic local convergence rate. These results are consistent with the convergence rate analysis and complexity analysis in Section 4.3.

COA v.s. DAWA: DAWA [11] targets very different applications compared to the proposed solution COA. In particular, DAWA focuses on range processing under exact (i.e., ϵ-) differential privacy, whereas COA addresses arbitrary linear counting queries under approximate (i.e., (ϵ, δ)-) differential privacy. Adapting DAWA to approximate differential privacy is non-trivial, because at the core of DAWA lies a dynamic programming algorithm that is specially designed for ℓ_1 cost and the Laplace mechanism (refer to Section 3.2 in [11]). Further, DAWA replies on certain assumptions of the underlying data, e.g., adjacent counts are similar in value, whereas COA is data-independent. Hence, their relative performance depends on the choice of parameter δ, as well as the dataset.

We compare COA with different values of δ ranging from 0.01 to 0.00001 against DAWA on workload *WRange*, since DAWA focuses on range queries. For brevity, we only present the experimental result on a real dataset (*Social Network*) and a synthetic one called *Random Uniform*. Specifically, the sensitive data *Random Uniform* consists of a random vector $\mathbf{x} \in \mathbb{R}^n$ drawn from the uniform distribution with mean zero and variance 10. Experimental comparisons of COA and DAWA on other datasets can be found in the full version [22]. Figure 14 shows the results with varying domain size n and the number of queries m. We have the following observations. (i) On the *Random Uniform* dataset, the performance of DAWA is rather poor, since this synthetic dataset does not satisfy the assumption that adjacent aggregates have similar values. (ii) COA generally achieves better performance than DAWA when $\delta \geq 0.0001$. (iii) With a fixed number of queries $m = 1024$, COA significantly outperforms DAWA when n is large. (iv) DAWA outperforms COA only when δ is very small, and the dataset happens to satisfy its assumptions. In such situations, one potential way to improve COA is to incorporate data-dependent information through a post-processing technique (e.g., [8, 10]), which is outside of the scope of this paper and left as future work.

6. CONCLUSION AND FUTURE WORK

In this paper we introduce a convex re-formulation for optimizing batch linear aggregate queries under approximate differential privacy. We provide a systematic analysis of the resulting convex optimization problem. In order to solve the convex problem, we propose a Newton-like method, which is guaranteed to achieve globally linear convergence rate and locally quadratic convergence rate. Extensive experiment on real world data sets demonstrate that our method is efficient and effective.

There are several interesting directions for future research. Firstly, it is worthwhile to extend the proposed method to develop hybrid data and workload aware differentially private algorithms [11, 10]. Secondly, it is interesting to investigate convex relaxations/reformulations to handle the squared/absolute sum error under differential privacy.

Acknowledgments

We would like to thank Prof. Shaohua Pan (SCUT) for her helpful discussions and the reviewers for their valuable comments. This research is partly supported by the research grant for the Human Centered Cyber physical Systems Programme at the Advanced Digital Sciences Center from Singapore's A*STAR. This research is also supported in part by Postdoctoral Science Foundation of China (2015M572317), Fundamental Research Funds for Central Universities, NSFC of Guangdong Joint Found (U1501254), NSF of China (61402182, 61472089 and 61572143), NSF of Guangdong (2014A030306004 and 2014A030308008), and Science and Technology Planning Project of Guangdong (2013B051000076, 2015B010108006 and 2015B010131015).

7. REFERENCES

[1] J. Dattorro. *Convex Optimization & Euclidean Distance Geometry*. Meboo Publishing USA, 2011.

[2] C. Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.

[3] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *EuroCRYPT*, pages 486–503, 2006.

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

[5] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics*, 32:407–499, 2002.

[6] Ú. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *CCS*, 2014.

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

[8] J. Lee, Y. Wang, and D. Kifer. Maximum likelihood postprocessing for differential privacy under consistency constraints. In *SIGKDD*, pages 635–644, 2015.

[9] J. D. Lee, Y. Sun, and M. A. Saunders. Proximal newton-type methods for minimizing composite functions. *SIAM Journal on Optimization*, 24(3):1420–1443, 2014.

[10] C. Li. Optimizing linear queries under differential privacy. *PhD Thesis, University of Massachusetts*, 2013.

[11] 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.

[12] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *PODS*, pages 123–134, 2010.

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

[14] C. Li and G. Miklau. Optimal error of query sets under the differentially-private matrix mechanism. In *ICDT*, pages 272–283, 2013.

[15] F. McSherry and I. Mironov. Differentially private recommender systems: Building privacy into the netflix prize contenders. In *SIGKDD*, pages 627–636, 2009.

[16] Y. Nesterov and A. Nemirovski. *Interior-point Polynomial Algorithms in Convex Programming*. Society for Industrial Mathematics, 1994.

[17] A. Nikolov, K. Talwar, and L. Zhang. The geometry of differential privacy: the sparse and approximate cases. In *STOC*, pages 351–360, 2013.

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

[19] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

[20] P. Tseng and S. Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117(1-2):387–423, 2009.

[21] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. In *ICDE*, pages 225–236, 2010.

[22] G. Yuan, Y. Yang, Z. Zhang, and Z. Hao. Convex optimization for linear query processing under approximate differential privacy. *arXiv preprint, url: http://arxiv.org/abs/1602.04302*, 2016.

[23] 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.

[24] G. Yuan, Z. Zhang, M. Winslett, X. Xiao, Y. Yang, and Z. Hao. Optimizing batch linear queries under exact and approximate differential privacy. *ACM Transactions on Database Systems*, 40(2):11, 2015.

[25] S. Yun, P. Tseng, and K. Toh. A block coordinate gradient descent method for regularized convex separable optimization and covariance selection. *Mathematical Programming*, 129(2):331–355, 2011.