A. Estimation

Define $\boldsymbol{\mu}(\boldsymbol{\theta})$ to be the marginals of the graphical model with parameters $\boldsymbol{\theta}$, which may be computed with the MARGINAL-ORACLE.

A.1. Proximal Algorithm Derivation

Our goal is to solve the following optimization problem:

$$\hat{\boldsymbol{\mu}} = \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathcal{M}} L(\boldsymbol{\mu})$$

where L is some convex function such as $\|\mathbf{Q}_{\mathcal{C}}\boldsymbol{\mu} - \mathbf{y}\|$.

Using the mirror descent algorithm (Beck & Teboulle, 2003), we can use the following update equation:

$$\boldsymbol{\mu}^{t+1} = \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathcal{M}} \boldsymbol{\mu}^T \nabla L(\boldsymbol{\mu}^t) + \frac{1}{\eta_t} D(\boldsymbol{\mu}, \boldsymbol{\mu}^t)$$

Where D is a Bregman distance measure defined as

$$D(\boldsymbol{\mu}, \boldsymbol{\mu}^t) = \psi(\boldsymbol{\mu}) - \psi(\boldsymbol{\mu}^t) - (\boldsymbol{\mu} - \boldsymbol{\mu}^t)^T \nabla \psi(\boldsymbol{\mu}^t)$$

for some strongly convex and continuously differentiable function ψ. Using $\psi = -H$ to be the negative entropy, we arrive at the following update equation:

$$\begin{aligned}
\boldsymbol{\mu}^{t+1} &= \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathcal{M}} \boldsymbol{\mu}^T \nabla L(\boldsymbol{\mu}^t) + \frac{1}{\eta_t} D(\boldsymbol{\mu}, \boldsymbol{\mu}^t) \\
&= \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathcal{M}} \boldsymbol{\mu}^T \nabla L(\boldsymbol{\mu}^t) + \frac{1}{\eta_t} \Big(- H(\boldsymbol{\mu}) + \boldsymbol{\mu}^T \nabla H(\boldsymbol{\mu}^t) \Big) \\
&= \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathcal{M}} \boldsymbol{\mu}^T \Big(\nabla L(\boldsymbol{\mu}^t) + \frac{1}{\eta_t} \nabla H(\boldsymbol{\mu}^t) \Big) - \frac{1}{\eta_t} H(\boldsymbol{\mu}) \\
&= \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathcal{M}} \boldsymbol{\mu}^T \Big(\eta_t \nabla L(\boldsymbol{\mu}^t) + \nabla H(\boldsymbol{\mu}^t) \Big) - H(\boldsymbol{\mu}) \\
&= \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathcal{M}} \boldsymbol{\mu}^T \Big(\eta_t \nabla L(\boldsymbol{\mu}^t) - \boldsymbol{\theta}^t \Big) - H(\boldsymbol{\mu}) \\
&= \boldsymbol{\mu}\big(\boldsymbol{\theta}^t - \eta_t \nabla L(\boldsymbol{\mu}^t) \big)
\end{aligned}$$

The first four steps are simple algebraic manipulation of the mirror descent update equation. The final two steps use the observation that $\nabla H(\boldsymbol{\mu}^t) = -\boldsymbol{\theta}^t$ and that marginal inference can be cast as the following optimization problem: (Wainwright & Jordan, 2008; Vilnis et al., 2015)

$$\boldsymbol{\mu}(\boldsymbol{\theta}) = \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathcal{M}} -\boldsymbol{\mu}^T \boldsymbol{\theta} - H(\boldsymbol{\mu})$$

Thus, optimization over the marginal polytope is reduced to computing the marginals of a graphical model with parameters $\boldsymbol{\theta}^t - \eta_t \nabla L(\boldsymbol{\mu}^t)$, which can be accomplished using belief propagation or some other MARGINAL-ORACLE.

A.2. Accelerated Proximal Algorithm Derivation

The derivation of the accelerated proximal algorithm is similar. It is based on Algorithm 3 from (Xiao, 2010). Applied to our setting, step 4 of that algorithm requires solving the following problem:

$$\begin{aligned}
\boldsymbol{\nu}^t &= \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathcal{M}} \boldsymbol{\mu}^T \bar{\boldsymbol{g}} - \frac{4K}{t(t+1)} H(\boldsymbol{\mu}) \\
&= \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathcal{M}} \frac{t(t+1)}{4K} \boldsymbol{\mu}^T \bar{\boldsymbol{g}} - H(\boldsymbol{\mu}) \\
&= \boldsymbol{\mu}\Big(- \frac{t(t+1)}{4L} \bar{\boldsymbol{g}} \Big)
\end{aligned}$$

which we solve by using the MARGINAL-ORACLE.

A.3. Direct Optimization

In preliminary experiments we also evaluated a direct method to solve the optimization problem. For the direct method, we estimate the parameters $\hat{\boldsymbol{\theta}}$ directly by reformulating the optimization problem and instead solving the unconstrained problem $\hat{\boldsymbol{\theta}} = \operatorname{argmin}_{\boldsymbol{\theta}} L\big(\boldsymbol{\mu}(\boldsymbol{\theta})\big)$.To evaluate the optimization objective, we use MARGINAL-ORACLE to compute $\boldsymbol{\mu}(\boldsymbol{\theta})$ and then compute the loss. For optimization, it has been observed that it is possible to back-propagate through marginal inference procedures (with or without automatic differentiation software) to compute their gradients (Eaton & Ghahramani, 2009; Domke, 2013). We apply automatic differentiation to the entire forward computation (Maclaurin et al., 2015), which includes MARGINAL-ORACLE, to compute the gradient of L.

Since this is now an unconstrained optimization problem and we can compute the gradient of L, many optimization methods apply. In our experiments, we use L_2 loss, which is smooth, and apply the L-BFGS algorithm for optimization (Byrd et al., 1995).

However, despite its simplicity, there is a significant drawback to the direct algorithm. It is not, in general, convex with respect to $\boldsymbol{\theta}$. This may seem surprising since the original problem is convex, i.e., $L(\boldsymbol{\mu})$ is convex with respect to $\boldsymbol{\mu}$ and $\mathcal{M}$ is convex. Also, the most well known problem of this form, maximum-likelihood estimation in graphical models, *is* convex with respect to $\boldsymbol{\theta}$ (Wainwright & Jordan, 2008); however, this relies on properties of exponential families that do not apply to other loss functions. One can verify for losses as simple as L_2 that the Hessian need not be positive definite. As a result, the direct algorithm is not guaranteed to converge to a global minimum of the original convex optimization problem $\min_{\boldsymbol{\mu} \in \mathcal{M}} L(\boldsymbol{\mu})$. We did not observe convergence problems in our experiments, but it was not better in practice than the proximal algorithms, which is why it is not included in the paper.

Algorithm 3 Inference for Factored Queries

Input: Parameters $\boldsymbol{\theta}$, factored query matrix $\mathbf{Q}$
Output: Query answers $\mathbf{Q}\,\mathbf{p}_{\boldsymbol{\theta}}$
$\psi = \{\exp(\boldsymbol{\theta}_C) \mid C \in \mathcal{C}\} \cup \{\mathbf{Q}_i \mid i \in [d]\}$
$Z =$MARGINAL-ORACLE$(\boldsymbol{\theta})$
return VARIABLE-ELIM$(\psi, \mathcal{X})/Z$

B. Inference

We now discuss how to exploit our compact factored representation of $\mathbf{p}_{\boldsymbol{\theta}}$ to answer new linear queries. We give an efficient algorithm for answering *factored linear queries*.

Definition 4 (Factored Query Matrix)**.** *A factored query matrix $\mathbf{Q}$ has columns that are indexed by $\mathbf{x}$ and rows that are indexed by vectors $\mathbf{z} \in [r_1] \times \cdots \times [r_d]$. The total number of rows (queries) is $r = \prod_{i=1}^{d} r_i$. The entries of $\mathbf{Q}$ are given by $\mathbf{Q}(\mathbf{z}, \mathbf{x}) = \prod_{i=1}^{d} \mathbf{Q}_i(\mathbf{z}_i, \mathbf{x}_i)$, where $\mathbf{Q}_i \in \mathbb{R}^{r_i \times n_i}$ is a specified factor for the ith attribute. The matrix $\mathbf{Q}$ can be expressed as $\mathbf{Q} = \mathbf{Q}_1 \otimes \cdots \otimes \mathbf{Q}_d$, where $\otimes$ is the Kronecker product.*

Factored query matrices are expressive enough to encode any conjunctive query (or a cartesian product of such queries), and more. There are a number of concrete examples that demonstrate the usefulness of answering queries of this form, including:

- Computing the marginal $\boldsymbol{\mu}_C$ for any $C \subseteq [d]$ (including unmeasured marginals).
- Computing the multivariate CDF of $\boldsymbol{\mu}_C$ for any $C \subseteq [d]$.
- Answering range queries.
- Compressing the distribution by transforming the domain.
- Computing the (unnormalized) expected value of one variable conditioned on other variables.

For the first two examples, we could have used standard variable elimination to eliminate all variables except those in C. Existing algorithms are not able to handle the other examples without materializing $\hat{\mathbf{p}}$ (or a marginal that supports the queries). Thus, our algorithm generalizes variable elimination. A more comprehensive set of examples, and details on how to construct these query matrices are given in section B.1

The procedure for answering these queries is given in Algorithm 3, which can be understood as follows. For a particular $\mathbf{z}$, write $f(\mathbf{z}, \mathbf{x}) = \mathbf{Q}(\mathbf{z}, \mathbf{x})\mathbf{p}_{\boldsymbol{\theta}}(\mathbf{x}) = \prod_i \mathbf{Q}_i(\mathbf{z}_i, \mathbf{x}_i)\mathbf{p}_{\boldsymbol{\theta}}(\mathbf{x})$. This can be viewed as an augmented graphical model on the variables $\mathbf{z}$ and $\mathbf{x}$ where we have introduced new pairwise factors between each $(\mathbf{x}_i, \mathbf{z}_i)$ pair defined by the query matrix. Unlike a regular graphical model, the new factors can contain negative values. The query answers are obtained by multiplying $\mathbf{Q}$ and $\mathbf{p}$, which sums over $\mathbf{x}$. The $\mathbf{z}$th answer is given by:

$$
\begin{aligned}
(\mathbf{Q}\mathbf{p}_{\boldsymbol{\theta}})(\mathbf{z}) &= \sum_{\mathbf{x} \in \mathcal{X}} \mathbf{Q}(\mathbf{z}, \mathbf{x})\mathbf{p}_{\boldsymbol{\theta}}(\mathbf{x}) \\
&= \frac{1}{Z} \sum_{\mathbf{x} \in \mathcal{X}} \prod_{i=1}^{d} \mathbf{Q}_i(\mathbf{z}_i, \mathbf{x}_i) \prod_{C \in \mathcal{C}} \exp[\boldsymbol{\theta}_C(\mathbf{x}_C)]
\end{aligned}
$$

This can be understood as marginalizing over the $\mathbf{x}$ variables in the augmented model $f(\mathbf{z}, \mathbf{x})$. The VARIABLE-ELIM routine referenced in the algorithm is standard variable elimination to perform this marginalization; it can handle negative values with no modification. We stress that, in practice, factor matrices $\mathbf{Q}_i$ may have only one row ($r_i = 1$, e.g., for marginalization); hence the output size $r = \prod_{i=1}^{d} r_i$ is not necessarily exponential in d.

B.1. Factored Query Matrices

Table 2 gives some example "building block" factors that can be used to construct factored query matrices. This is by no means an exhaustive list of possible factors but it provides the reader with evidence that answering these types of queries efficiently is practically useful. The factored query matrix for computing the marginal $\boldsymbol{\mu}_C$ uses $\mathbf{Q}_i = \mathbf{I}$ for $i \in C$ and $\mathbf{Q}_i = \mathbf{1}$ for $i \notin C$. Similarly, the factored query matrix for computing the multivariate CDF of $\boldsymbol{\mu}_C$ would simply use $\mathbf{Q}_i = \mathbf{P}$ for $i \in C$. A query matrix for compressing a distribution could be characterized by functions $f_i : [n_i] \to [2]$ or equivalently binary matrices $\mathbf{Q}_i = \mathbf{R}_{f_i} \in \mathbb{R}^{2 \times n_i}$. The query matrix for computing the (unnormalized) expected value of variable i conditioned on variable j would use $\mathbf{Q}_i = \mathbf{E}$ and $\mathbf{Q}_j = \mathbf{I}$ (and $\mathbf{Q}_k = \mathbf{1}$ for all other k). These are only a few examples; these building blocks can be combined arbitrarily to construct a wide variety of interesting query matrices.

C. Loss Functions

C.1. L_1 and L_2 losses

The L_1 and L_2 loss functions have simple (sub)gradients.

$$
\begin{aligned}
\nabla L_1(\boldsymbol{\mu}) &= \mathbf{Q}_{\mathcal{C}}^T \text{sign}(\mathbf{Q}_{\mathcal{C}}\boldsymbol{\mu} - \mathbf{y}) \\
\nabla L_2(\boldsymbol{\mu}) &= \mathbf{Q}_{\mathcal{C}}^T (\mathbf{Q}_{\mathcal{C}}\boldsymbol{\mu} - \mathbf{y})
\end{aligned}
$$

C.2. Linear measurements with unequal noise

When the privacy budget is not distributed evenly to the measurements in the we have to appropriately modify the loss functions, which assume that the noisy answers all have equal variance. In order to do proper estimation and inference we have to account for this varying noise level in the loss function. In section 3.1 we claimed that $L(\mathbf{p}) = \|\mathbf{Q}\mathbf{p} - \mathbf{y}\|$ makes sense as a loss function when the noise introduced to $\mathbf{y}$ are iid. Luckily, even if this assumption is

$\mathbf{Q}_i$	Requirements	Size	Definition ($\forall a \in [n_i]$)	Description
$\mathbf{I}$		$n_i \times n_i$	$\mathbf{Q}_i(a,a) = 1$	keep variable in
$\mathbf{1}$		$1 \times n_i$	$\mathbf{Q}_i(1,a) = 1$	marginalize variable out
$\boldsymbol{e}_j$	$j \in [n_i]$	$1 \times n_i$	$\mathbf{Q}_i(1,j) = 1$	inject evidence
$\boldsymbol{e}_S$	$S \subseteq [n_i]$	$1 \times n_i$	$\mathbf{Q}_i(1,j) = 1\ \forall j \in S$	inject evidence (disjuncts)
$\mathbf{P}$		$n_i \times n_i$	$\mathbf{Q}_i(b,a) = 1\ \forall b \geq a$	transform into CDF
$\mathbf{R}_f$	$f : [n_i] \to [r_i]$	$r_i \times n_i$	$\mathbf{Q}_i(f(a),a) = 1$	compress domain
$\mathbf{E}$		$1 \times n_i$	$\mathbf{Q}_i(1,a) = a$	reduce to mean
$\mathbf{E}_k$	$k \geq 1$	$k \times n_i$	$\mathbf{Q}_i(b,a) = a^b\ \forall b \leq k$	reduce to first k moments

Table 2: Example factors in the factored query matrix

not satisfied it is easy to correct. Assume that $y_i = \mathbf{q}_i^T \mathbf{p} + \varepsilon_i$ where $\varepsilon_i \sim Lap(b_i)$. Then $\frac{1}{b_i} y_i = \frac{1}{b_i} \mathbf{q}_i^T \mathbf{p} + \frac{1}{b_i} \varepsilon_i$ and $\frac{1}{b_i} \varepsilon_i \sim Lap(1)$. Thus, we can replace the query matrix $\mathbf{Q} \leftarrow \mathbf{D}\mathbf{Q}$ and the answer vector $\mathbf{y} \leftarrow \mathbf{D}\mathbf{y}$ where $\mathbf{D}$ is the diagonal matrix defined by $\mathbf{D}_{ii} = \frac{1}{b_i}$. All the new query answers have the same effective noise scale, and so the standard loss functions may be used. This idea still applies if the noise on each query answer is sampled from a normal distribution as well (for (ϵ, δ)-differential privacy).

C.3. Dual Query Loss Function

Algorithm 4 shows DualQuery applied to workloads defined over the marginals of the data. There are five hyper-parameters, of which four must be specified and the remaining one can be determined from the others.

The first step of the algorithm computes the answers to the workload queries. Then for T time steps observations are made about the true data via samples from the distribution Q^t. These observations are used to find a record $\mathbf{x} \in \mathcal{X}$ to add to the synthetic database.

Algorithm 4 Dual Query for marginals workloads

Input: $\boldsymbol{X}$, the true data
Input: $\mathbf{W}_{\mathcal{C}}$, workload queries
Input: $(s, T, \eta, \epsilon, \delta)$, hyper-parameters
Output: synthetic database of T records
$\mathbf{y} = \mathbf{W}_C \boldsymbol{\mu}_{\boldsymbol{X}}$
$Q^1 = \text{uniform}(\mathbf{W})$
for $t = 1, \ldots, T$ **do**
  sample $\mathbf{q}_1^t, \ldots \mathbf{q}_s^t$ from Q^t
  $\mathbf{x}^t = \text{argmax}_{\mathbf{x} \in \mathcal{X}} \sum_{i=1}^s \mathbf{q}_i^t \boldsymbol{\mu} - \mathbf{q}_i^t \boldsymbol{\mu}_{\mathbf{x}}$
  $Q^{t+1} = Q^t \odot \exp\left(-\eta * (\mathbf{y} - \mathbf{W}_C \boldsymbol{\mu}_{\mathbf{x}^t})\right)$
  normalize Q^t
end for
return $(\mathbf{x}^1, \ldots, \mathbf{x}^T)$

Algorithm 5 shows a procedure for computing the negative log likelihood (our loss function) of observing the DualQuery output, given some marginals. Evaluating the log likelihood is fairly expensive, as it requires basically simulating the entire DualQuery algorithm. Fortunately we do not have to run the most computationally expensive step within the procedure, which is finding $\mathbf{x}^t$. We differentiate this loss function using automatic differentiation (Maclaurin et al., 2015) for use within our estimation algorithms.

Algorithm 5 Dual Query Loss Function

Input: $\boldsymbol{\mu}$, marginals of the data
Input: $\mathbf{W}_{\mathcal{C}}$, workload queries
Input: cache, all relevant output from DualQuery
> $\mathbf{q}_1^t, \ldots \mathbf{q}_s^t$ - sampled queries at each time step
> $\mathbf{x}^t$ - chosen record at each time step
Output: $L(\boldsymbol{\mu})$, the negative log likelihood
$\mathbf{y} = \mathbf{W}_C \boldsymbol{\mu}$
$Q^1 = \text{uniform}(\mathbf{W})$
loss $= 0$
for $t = 1, \ldots, T$ **do**
  loss $-= \sum_{i=1}^s \log\left(Q^t(\mathbf{q}_i^t)\right)$
  $Q^{t+1} = Q^t \odot \exp\left(-\eta * (\mathbf{y} - \mathbf{W}_C \boldsymbol{\mu}_{\mathbf{x}^t})\right)$
  normalize Q^t
end for
return loss

D. Additional Experiments

D.1. L_1 vs. L_2 Loss

In Section 3 we mentioned that minimizing L_1 loss is equivalent to maximizing likelihood for linear measurements with Laplace noise, but that L_2 loss is more commonly used in the literature. In this experiment we compare these two estimators side-by-side. Specifically, we consider the workload from Figure 1 and measurements chosen by HDMM with $\epsilon = 1.0$. As expected, performing L_1 minimization results in lower L_1 loss but higher L_2 loss, although the difference is quite small, especially for L_1 loss. The difference is larger for L_2 loss. Minimizing L_2 loss results in lower workload error, indicating that it generalizes better. This is somewhat surprising given that L_1 minimization is maximizing likelihood. Another interesting observation is

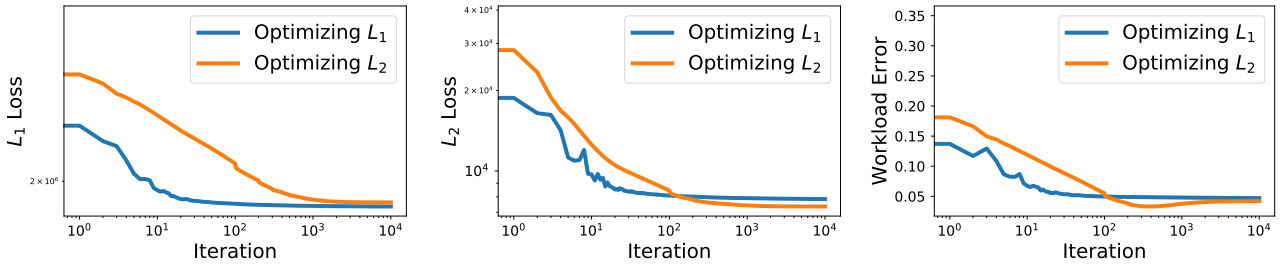

Figure 3: L_1 minimization vs. L_2 minimization, evaluated on L_1 loss, L_2 loss, and workload error

that the workload error actually starts going up after about 200 iterations, suggesting that some form of over-fitting is occurring. There minimum workload error achieved was 0.066 while the final workload error was 0.084 — a pretty meaningful difference. Of course, in practice we cannot stop iterating when workload error starts increasing because evaluating it requires looking at the true data.

E. Additional Details

E.1. Unknown Total

Our algorithms require m, the total number of records in the dataset is known or can be estimated. Under a slightly different privacy definition where $\text{nbrs}(\mathbf{X})$ is the set of databases where a single record is added or removed (instead of modified), this total is a sensitive quantity which cannot be released exactly (Dwork & Roth, 2014). Thus, the total is not known in this setting, but a good estimate can typically be obtained from the measurements taken, without spending additional privacy budget. First observe that $\mathbf{1}^T\boldsymbol{\mu}_C = m$ is the total for an unnormalized database. Now suppose we have measured $\mathbf{y}_C = \mathbf{Q}_C\boldsymbol{\mu}_C + \mathbf{z}_C$. Then as long as $\mathbf{1}^T$ is in the row-space of $\mathbf{Q}_C$, $m_C = \mathbf{1}^T\mathbf{Q}_C^+\mathbf{y}_C$ is an unbiased estimate for m with variance $Var(m_C) = Var(\mathbf{y}_C)\left\|\mathbf{1}^T\mathbf{Q}_C^+\right\|_2^2$. This is a direct consequence of Proposition 9 from (Li et al., 2015). We thus have multiple estimates for m which we can combine using inverse variance weighting, resulting in the final estimate of $\hat{m} = \frac{\sum_C m_C/Var(m_C)}{\sum_C 1/Var(m_C)}$, which we can use in place of m.

E.2. Multiplicative Weights vs Entropic Mirror Descent

Recall from Section 4 that the multiplicative weights update equation is:

$$\hat{\mathbf{p}} \leftarrow \hat{\mathbf{p}} \odot \exp\left(\mathbf{q}_i(\mathbf{q}_i^T\hat{\mathbf{p}} - y_i)\right)/2m/Z$$

and the update is applied (possibly cyclically) for $i = 1, \ldots, T$. Now imagine taking all of the measurements and organizing them into a $T \times n$ matrix $\mathbf{Q}$. Then we can apply all the updates at once, instead of sequentially, and we end up with the following update equation.

$$\hat{\mathbf{p}} \leftarrow \hat{\mathbf{p}} \odot \exp\left(\mathbf{Q}^T(\mathbf{Q}\hat{\mathbf{p}} - \mathbf{y})/2m\right)/Z$$

Observing that $\nabla L_2(\hat{\mathbf{p}}) = \mathbf{Q}^T(\mathbf{Q}\hat{\mathbf{p}} - \mathbf{y})$, this simplifies to:

$$\hat{\mathbf{p}} \leftarrow \hat{\mathbf{p}} \odot \exp\left(\nabla L_2(\hat{\mathbf{p}})/2m\right)/Z$$

which is precisely the update equation for entropic mirror descent for minimizing $L_2(\mathbf{p})$ over the probability simplex (Beck & Teboulle, 2003).