EntropyDB: A Probabilistic Approach to Approximate Query Processing

Laurel Orr, Magdalena Balazinska, and Dan Suciu
University of Washington
Seattle, Washington, USA

{ljorr1, magda, suciu}@cs.washington.edu

ABSTRACT

We present EntropyDB[1], an interactive data exploration system that uses a probabilistic approach to generate a small, query-able summary of a dataset. Departing from traditional summarization techniques, we use the Principle of Maximum Entropy to generate a probabilistic representation of the data that can be used to give approximate query answers. We develop the theoretical framework and formulation of our probabilistic representation and show how to use it to answer queries. We then present solving techniques, give two critical optimizations to improve preprocessing time and query execution time, and explore methods to reduce query error. Lastly, we experimentally evaluate our work using a 5 GB dataset of flights within the United States and a 210 GB dataset from an astronomy particle simulation. While our current work only supports linear queries, we show that our technique can successfully answer queries faster than sampling while introducing, on average, no more error than sampling and can better distinguish between rare and nonexistent values. We also discuss extensions that can allow for data updates and linear queries over joins.

1. INTRODUCTION

Interactive data exploration allows a data analyst to browse, query, transform, and visualize data at "human speed" [17]. It has been long recognized that general-purpose DBMSs are ill suited for interactive exploration [36]. While users require interactive responses, they do not necessarily require precise responses because either the response is used in some visualization, which has limited resolution, or an approximate result is sufficient and can be followed

---

[1]This is an extended version of the VLDB 2017 paper "Probabilistic Database Summarization for Interactive Data Exploration" [38].

up with a more accurate, costly query if needed. *Approximate Query Processing* (AQP) refers to a set of techniques designed to allow fast but approximate answers to queries. All successful AQP systems to date rely on sampling or a combination of sampling and indices. The sample can either be computed on-the-fly, e.g., in the highly influential work on *online aggregation* [24] or systems like DBO [26] and Quickr [29], or precomputed offline, like in BlinkDB [4] or Sample+Seek [20]. Samples have the advantage that they are easy to compute, can accurately estimate aggregate values, and are good at detecting heavy hitters. However, sampling may fail to return estimates for small populations; targeted stratified samples can alleviate this shortcoming, but stratified samples need to be precomputed to target a specific query, defeating the original purpose of AQP.

In this paper, we propose an alternative approach to interactive data exploration based on the Maximum Entropy principle (MaxEnt). The MaxEnt model has been applied in many settings beyond data exploration; e.g., the *multiplicative weights* mechanism [23] is a MaxEnt model for both differentially private and, by [21], statistically valid answers to queries, and it has been shown to be theoretically optimal. In our setting of the MaxEnt model, the data is preprocessed to compute a probabilistic model. Then, queries are answered by doing probabilistic inference on this model. The model is defined as the probabilistic space that obeys some observed statistics on the data and makes no other assumptions (Occam's principle). The choice of statistics boils down to a precision/memory tradeoff: the more statistics one includes, the more precise the model and the more space required. Once computed, the MaxEnt model defines a probability distribution on possible worlds, and users can interact with this model to obtain approximate query results. Unlike a sample, which may miss rare items, the MaxEnt model can infer something about every query.

Despite its theoretical appeal, the computational challenges associated with the MaxEnt model make it difficult to use in practice. In this paper, we develop the first scalable techniques to compute and use the MaxEnt model. As an application, we illustrate it with interactive data exploration. Our first contribution is to simplify the standard MaxEnt model to a form that is appropriate for data summarization (Sec. 3). We show how to simplify the MaxEnt model to be a multi-linear polynomial that has one monomial for each possible tuple (Sec. 3, Eq. 6) rather than its naïve form that has one monomial for each possible world (Sec. 2, Eq. 2). Even with this simplification, the MaxEnt model starts by being larger than the data. For example,

our smaller experimental dataset (introduced in Sec. 7) is 5 GB, but the number of possible tuples is approximately 10^{10}, which is 74 GB if each tuple is 8 bytes. Our *first optimization* consists of a compression technique for the polynomial of the MaxEnt model (Sec. 4.1); for example, for our smaller experimental dataset, the summary is below 200MB, while for our larger dataset of 210GB, it is less than 1GB. Our *second optimization* consists of a new technique for query evaluation on the MaxEnt model (Sec. 4.2) that only requires setting some variables to 0; this reduces the runtime to be on average below 500ms and always below 1 second.

As mentioned above, there is a precision/memory tradeoff when choosing which statistics to use to define the model. To alleviate this problem, our *third optimization* develops a statistic selection technique based on K-D trees that groups together individual statistics of similar value and uses the single group statistic in the model rather than the individual ones. We also explore optimal sorting techniques to encourage similar values to be clustered together before building our K-D trees.

We find that the main bottleneck in using the MaxEnt model is computing the model itself; in other words, computing the values of the variables of the polynomial such that it matches the existing statistics over the data. Solving the MaxEnt model is difficult; prior work for multi-dimensional histograms [35] uses an iterative scaling algorithm for this purpose. To date, it is well understood that the MaxEnt model can be solved by reducing it to a convex optimization problem [47] of a *dual* function (Sec. 2), which can be solved using Gradient Descent. However, even this is difficult given the size of our model. We managed to adapt a variant of Stochastic Gradient Descent called Mirror Descent [11], and our optimized query evaluation technique can compute the MaxEnt model for large datasets in under a day.

Lastly, to expand on how the MaxEnt model can be used in a full-fledged database system, we discuss handing data updates and answering join queries using the MaxEnt model. We also elaborate on the connection between the MaxEnt model and graphical models.

In summary, in this paper, we develop the following new techniques (the asterisked items are new developments from [38]):

- A closed-form representation of the probability space of possible worlds using the Principle of Maximum Entropy, and a method to use the representation to answer queries in expectation (Sec. 3).
- A compression technique and optimized implementation* of the compression for the MaxEnt summary (Sec. 4.1, Sec. 5.1).
- Optimized query processing techniques, including implementation details* (Sec. 4.2, Sec. 5.2).
- A new method for selecting 2-dimensional statistics based on optimal matrix reorderings* and a modified K-D tree (Sec. 6)
- Detailed experiments comparing the accuracy of the MaxEnt summary versus various sampling techniques (Sec. 7.3).
- Solving time* and query runtime evaluations showing our interactive query speeds (Sec. 7.4.1, Sec. 7.4.2).
- A discussion on how the MaxEnt summary relates to probabilistic databases and graphical models* (Sec. 8).

- A description on how the MaxEnt summary can be extended to handle data updates and joins* (Sec. 8.2).

We implement the above techniques in a prototype system that we call EntropyDB and evaluate it on the flights and astronomy datasets. We find that EntropyDB can answer queries faster than sampling while introducing no more error, on average, and does better at identifying small populations.

2. BACKGROUND

We summarize data by fitting a probability distribution over the active domain. The distribution assumes that the domain values are distributed in a way that preserves given statistics over the data but are otherwise uniform.

For example, consider a data scientist who analyzes a dataset of flights in the United States for the month of December 2013. All she knows is that the dataset includes all flights within the 50 possible states and that there are 500,000 flights in total. She wants to know how many of those flights are from CA to NY. Without any extra information, our approach would assume all flights are equally likely and estimate that there are $500,000/50^2 = 200$ flights.

Now suppose the data scientist finds out that flights leaving CA only go to NY, FL, or WA. This changes the estimate because instead of there being $500,000/50 = 10,000$ flights leaving CA and uniformly going to all 50 states, those flights are only going to 3 states. Therefore, the estimate becomes $10,000/3 = 3,333$ flights.

This example demonstrates how our summarization technique takes into account these existing statistics over flights going to and from specific states to answer queries, and the rest of this section covers its theoretical foundation.

2.1 Possible World Semantics

To model a probabilistic database, we use a slotted possible world semantics where rows have an inherent unique identifier, meaning the order of the tuples matters. Our set of possible worlds is generated from the active domain and size of each relation. Each database instance is one possible world with an associated probability such that the probabilities of all possible worlds sum to one.

In contrast to typical probabilistic databases where a relation is tuple-independent and the probability of a relation is calculated from the product of the probability of each tuple, we calculate a relation's probability from a formula derived from the MaxEnt principle and a set of constraints on the overall distribution[2]. This approach captures the idea that the distribution should be uniform except where otherwise specified by the given constraints.

2.2 The Principle of Maximum Entropy

The Principle of Maximum Entropy (MaxEnt) states that subject to prior data, the probability distribution which best represents the state of knowledge is the one that has the largest entropy. This means given our set of possible worlds,

---

[2]Using the MaxEnt principle will generate a probability distribution that is different from the tuple-independent distribution because the MaxEnt principle does not guarantee tuple independence.

PWD, the probability distribution $\Pr(I)$ is one that agrees with the prior information on the data and maximizes

$$-\sum_{I \in PWD} \Pr(I) \log(\Pr(I))$$

where I is a database instance, also called possible world. The above probability must be normalized, $\sum_I \Pr(I) = 1$, and must satisfy the prior information represented by a set of k expected value constraints:

$$s_j = \mathbb{E}[\phi_j(I)], \quad j = 1, k \qquad (1)$$

where s_j is a known value and ϕ_j is a function on I that returns a numerical value in $\mathbb{R}$. One example constraint is that the number of flights from CA to WI is 0.

Following prior work on the MaxEnt principle and solving constrained optimization problems [10, 47, 42], the MaxEnt probability distribution takes the form

$$\Pr(I) = \frac{1}{Z} \exp\left(\sum_{j=1}^{k} \theta_j \phi_j(I)\right) \qquad (2)$$

where θ_j is a parameter and Z is the following normalization constant:

$$Z \stackrel{\text{def}}{=} \sum_{I \in PWD} \left(\exp\left(\sum_{j=1}^{k} \theta_j \phi_j(I)\right)\right).$$

To compute the k parameters θ_j, we must solve the nonlinear system of k equations, Eq. 1, which is computationally difficult. However, it turns out [47] that Eq. 1 is equivalent to $\partial \Psi / \partial \theta_j = 0$ where the *dual* Ψ is defined as:

$$\Psi \stackrel{\text{def}}{=} \sum_{j=1}^{k} s_j \theta_j - \ln(Z).$$

Furthermore, Ψ is concave, which means solving for the k parameters can be achieved by maximizing Ψ. We note that Z is called the *partition function*, and its log, $\ln(Z)$, is called the *cumulant*.

Lastly, we adopt a slightly different notation where instead of e^{θ} we use α. Eq. 2 now becomes

$$\Pr(I) = \frac{1}{Z} \prod_{j=1}^{k} \alpha_j^{\phi_j(I)}. \qquad (3)$$

3. EntropyDB APPROACH

This section explains how we use the MaxEnt model for approximate query answering. We first show how we use the MaxEnt framework to transform a single relation R into a probability distribution represented by P. We then explain how we use P to answer queries over R. For reference, Table 1 lists common symbols and their definitions, and Table 2 lists various assumptions we incrementally make on our model and the section they are introduced.

3.1 Maximum Entropy Model of Data

We consider a single relation with m attributes and schema $R(A_1, \ldots, A_m)$ where each attribute, A_i, has an active domain D_i, assumed to be discrete and ordered.[3] Let $Tup = D_1 \times D_2 \times \cdots \times D_m = \{t_1, \ldots, t_d\}$ be the

---

[3]We support continuous data types by bucketizing their active domains.

m	# attrs
k	# statistics
D_i	domain of A_i
N_i	$|D_i|$
$\mathbf{q}$	linear query
$\mathbf{c}_j$	jth statistic query
π_j	jth statistic predicate
θ_j	$(\mathbf{c}_j, s_j)$
s_j	jth statistic constraint
ρ_{ij}	projection π_j onto A_i
$J_i \subseteq [k]$	1-dim statistic indices
$\mathcal{I} \subseteq [m]$	attr indices
$J_{\mathcal{I}} \subseteq [k]$	multi-dimensional statistic indices
B_a	# multi-dimensional attr sets
B_s	# stats per multi-dimensional attr set

Table 1: Common notation.

Queries are limited to linear queries.	Sec. 3.1
Continuous attributes are discretized.	Sec. 3.1
Summary includes all 1-dimensional statistics.	Sec. 3.1
Statistics are collections of range predicates.	Sec. 4.1
Each set of 2D statistics is disjoint.	Sec. 4.1
Summary adds only 2D high order statistics.	Sec. 6.1

Table 2: Model assumptions, and the section they are introduced.

set of all possible tuples. Denoting $N_i = |D_i|$, we have $d = |Tup| = \prod_{i=1}^{m} N_i$.

An *instance* for R is an ordered bag of n tuples, denoted I. For each I, we form a frequency vector which is a d-dimensional vector[4] $\mathbf{n}^I = [n_1^I, \ldots, n_d^I] \in \mathbb{R}^d$, where each number n_i^I represents the count of the tuple $t_i \in Tup$ in I (Fig. 1). The mapping from I to $\mathbf{n}^I$ is not one-to-one because the instance I is ordered, and two distinct instances may have the same counts. Further, for any instance I of cardinality n, $||\mathbf{n}^I||_1 = \sum_i n_i^I = n$. The frequency vector of an instance consisting of a single tuple $\{t_i\}$ is denoted $\mathbf{n}^{t_i} = [0, \ldots, 0, 1, 0, \ldots, 0]$ with a single value 1 in the ith position; i.e., $\{\mathbf{n}^{t_i} : i = 1, d\}$ forms a basis for $\mathbb{R}^d$.

While the MaxEnt principle allows us, theoretically, to answer any query probabilistically by averaging the query over all possible instances; in this paper, we limit our main analysis to linear queries but do discuss how to handle joins in Sec. 8.2. A *linear query* is a d-dimensional vector $\mathbf{q} = [q_1, \ldots, q_d]$ in $\mathbb{R}^d$. The answer to $\mathbf{q}$ on instance I is the dot product $\langle \mathbf{q}, \mathbf{n}^I \rangle = \sum_{i=1}^{d} q_i n_i^I$. With some abuse of notation, we will write $\mathbf{I}$ when referring to $\mathbf{n}^I$ and $\mathbf{t}_i$ when referring to $\mathbf{n}^{t_i}$. Notice that $\langle \mathbf{q}, \mathbf{t}_i \rangle = q_i$, and, for any instance I, $\langle \mathbf{q}, \mathbf{I} \rangle = \sum_i n_i^I \langle \mathbf{q}, \mathbf{t}_i \rangle$.

Fig. 1 illustrates the data and query model. Any counting query is a vector $\mathbf{q}$ where all coordinates are 0 or 1 and can be equivalently defined by a predicate π such that $\langle \mathbf{q}, \mathbf{I} \rangle = |\sigma_\pi(I)|$; with more abuse, we will use π instead of $\mathbf{q}$ when referring to a counting query. Other SQL queries can be modeled using linear queries, too. For example,

```
SELECT A, COUNT(*)
FROM R GROUP BY A
ORDER BY COUNT(*) DESC LIMIT 10
```

corresponds to several linear queries, one for each group, where the outputs are sorted and the top 10 returned.

Our goal is to compute a summary of the data that is small yet allows us to approximatively compute the answer

---

[4]This is a standard data model in several applications, such as differential privacy [31].

Domains:

$D_1 = \{a_1, a_2\}$ $N_1 = 2$

$D_2 = \{b_1, b_2\}$ $N_2 = 2$

$Tup = \{(a_1, b_1), (a_1, b_2), (a_2, b_1), (a_2, b_2)\}$ $d = 4$

Database Instance: Query:

I:	A	B	
1	a_1	b_1	Tup_1
2	a_1	b_2	Tup_2
3	a_2	b_2	Tup_4
4	a_1	b_1	Tup_1
5	a_2	b_2	Tup_4

q: `SELECT COUNT(*)`
     `FROM R`
     `WHERE A = a1`

Modeling Data and Query: n = 5, m = 2

$\mathbf{n}^{\mathbf{I}} = (2, 1, 0, 2)$ $\mathbf{q} = (1, 1, 0, 0)$ $\langle \mathbf{q}, \mathbf{n}^{\mathbf{I}} \rangle = 3$ also denoted $\langle \mathbf{q}, \mathbf{I} \rangle$

Figure 1: Illustration of the data and query model.

to any linear query. We assume that the cardinality n of R is fixed and known. In addition, we know k statistics, $\Phi = \{(\mathbf{c}_j, s_j) : j = 1, k\}$, where $\mathbf{c}_j$ is a linear query and $s_j \geq 0$ is a number. Intuitively, the statistic $(\mathbf{c}_j, s_j)$ asserts that $\langle \mathbf{c}_j, I \rangle = s_j$. For example, we can write 1-dimensional and 2-dimensional (2D) statistics like $|\sigma_{A_1=63}(I)| = 20$ and $|\sigma_{A_1 \in [50,99] \wedge A_2 \in [1,9]}(I)| = 300$.

Next, we derive the MaxEnt distribution for the possible instances I of a fixed size n. We replace the exponential parameters θ_j with $\ln(\alpha_j)$ so that Eq. 3 becomes

$$\Pr(I) = \frac{1}{Z} \prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{I} \rangle}. \tag{4}$$

We prove the following about the structure of the partition function Z:

LEMMA 3.1. *The partition function is given by*

$$Z = P^n \tag{5}$$

where P is the multi-linear polynomial

$$P(\alpha_1, \ldots, \alpha_k) \stackrel{def}{=} \sum_{i=1,d} \prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{t}_i \rangle}. \tag{6}$$

PROOF. Fix any $\mathbf{n} = [n_1, \ldots, n_d]$ such that $||\mathbf{n}||_1 = \sum_{i=1}^{d} n_i = n$. The number of instances I of cardinality n with $\mathbf{I} = \mathbf{n}$ is $n!/\prod_i n_i!$. Furthermore, for each such instance, $\langle \mathbf{c}_j, \mathbf{I} \rangle = \langle \mathbf{c}_j, \mathbf{n} \rangle = \sum_i n_i \langle \mathbf{c}_j, \mathbf{t}_i \rangle$. Therefore,

$$Z = \sum_I \Pr(I) = \sum_{\mathbf{n}: ||\mathbf{n}||_1 = n} \frac{n!}{\prod_i n_i!} \prod_{j=1,k} \alpha_j^{\sum_i n_i \langle \mathbf{c}_j, \mathbf{t}_i \rangle}$$

$$= \left(\sum_{i=1,d} \prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{t}_i \rangle} \right)^n = P^n.$$

$\square$

This restructuring of the partition function is valid because we represent an instance as an ordered bag rather than an unordered one.

The *data summary*, denoted $(P, \{\alpha_j\}, \Phi)$, consists of the polynomial P (Eq. 6), the values of its parameters α_j, and the statistics Φ. The statistics must be included as the polynomial parameters are defined by the linear queries $\mathbf{c}_j$ in the statistics Φ.

EXAMPLE 3.2. *Consider a relation with three attributes $R(A, B, C)$, and assume that the domain of each attribute has 2 distinct elements. Assume $n = 10$ and the only statistics in Φ are the following 1-dimensional statistics:*

$$(A = a_1, \; 3) \quad (B = b_1, \; 8) \quad (C = c_1, \; 6)$$
$$(A = a_2, \; 7) \quad (B = b_2, \; 2) \quad (C = c_2, \; 4).$$

The first statistic asserts that $|\sigma_{A=a_1}(I)| = 3$, etc. The polynomial P is

$$P = \alpha_1 \beta_1 \gamma_1 + \alpha_1 \beta_1 \gamma_2 + \alpha_1 \beta_2 \gamma_1 + \alpha_1 \beta_2 \gamma_2 +$$
$$\alpha_2 \beta_1 \gamma_1 + \alpha_2 \beta_1 \gamma_2 + \alpha_2 \beta_2 \gamma_1 + \alpha_2 \beta_2 \gamma_2$$

where α_1, α_2 are variables associated with the statistics on A, β_1, β_2 are for B[5], and γ_1, γ_2 are for C.

Consider the concrete instance that satisfies the above statistics

$$I = \{(a_1, b_2, c_2), (a_1, b_1, c_2), (a_1, b_1, c_2), (a_2, b_2, c_1)$$
$$(a_2, b_1, c_1), (a_2, b_1, c_1), (a_2, b_1, c_1), (a_2, b_1, c_1)\}.$$

Then, $\Pr(I) = \alpha_1^3 \alpha_2^7 \beta_1^8 \beta_2^2 \gamma_1^6 \gamma_2^4 / P^{10}$ where α_1^3 represents α_1 raised to the third power, α_2^7 represents α_2 to the seventh power, and so on.

EXAMPLE 3.3. *Continuing the previous example, we add the following multi-dimensional statistics to Φ:*

$$(A = a_1 \wedge B = b_1, \; 2) \quad (B = b_1 \wedge C = c_1, \; 5)$$
$$(A = a_2 \wedge B = b_2, \; 1) \quad (B = b_2 \wedge C = c_1, \; 1).$$

P is now

$$P = \alpha_1 \beta_1 \gamma_1 [\alpha\beta]_{1,1} [\beta\gamma]_{1,1} + \alpha_1 \beta_1 \gamma_2 [\alpha\beta]_{1,1} +$$
$$\alpha_1 \beta_2 \gamma_1 [\beta\gamma]_{2,1} + \alpha_1 \beta_2 \gamma_2 +$$
$$\alpha_2 \beta_1 \gamma_1 [\beta\gamma]_{1,1} + \alpha_2 \beta_1 \gamma_2 +$$
$$\alpha_2 \beta_2 \gamma_1 [\alpha\beta]_{2,2} [\beta\gamma]_{2,1} + \alpha_2 \beta_2 \gamma_2 [\alpha\beta]_{2,2}. \tag{7}$$

The red variables are the added 2-dimensional statistic variables; we use $[\alpha\beta]_{1,1}$ to denote a single variable corresponding to a 2-dimensional statistics on the attributes AB. Notice that each red variable only occurs with its related 1-dimensional variables. $[\alpha\beta]_{1,1}$, for example, is only in the same term as α_1 and β_1.

Now consider the earlier instance I. Its probability becomes $\Pr(I) = \alpha_1^3 \alpha_2^7 \beta_1^8 \beta_2^2 \gamma_1^6 \gamma_2^4 [\alpha\beta]_{1,1}^2 [\alpha\beta]_{2,2}^1 [\beta\gamma]_{1,1}^5 [\beta\gamma]_{2,1}^1 / P^{10}$.

To facilitate analytical queries, we choose the set of statistics Φ as follows:

- Each statistic $\phi_j = (\mathbf{c}_j, s_j)$ is associated with some predicate π_j such that $\langle \mathbf{c}_j, \mathbf{I} \rangle = |\sigma_{\pi_j}(I)|$. It follows that for every tuple t_i, $\langle \mathbf{c}_j, \mathbf{t}_i \rangle$ is either 0 or 1; therefore, each variable α_j has degree 1 in the polynomial P in Eq. 6.

- For each domain D_i, we include a complete set of 1-dimensional statistics in our summary. In other words, for each $v \in D_i$, Φ contains one statistic with predicate $A_i = v$. We denote $J_i \subseteq [k]$ the set of indices of the 1-dimensional statistics associated with D_i; therefore, $|J_i| = |D_i| = N_i$.

- We allow multi-dimensional statistics to be given by arbitrary predicates. They may be overlapping and/or incomplete; e.g., one statistic may count the tuples satisfying $A_1 \in [10, 30] \wedge A_2 = 5$ and another count the tuples satisfying $A_2 \in [20, 40] \wedge A_4 = 20$.

---

[5] We abuse notation here for readability. Technically, $\alpha_i = \alpha_{a_i}$, $\beta_i = \alpha_{b_i}$, and $\gamma_i = \alpha_{c_i}$.

- We assume the number of 1-dimensional statistics dominates the number of attribute combinations; i.e., $\sum_{i=1}^{m} N_i \gg 2^m$.
- If some domain D_i is large, it is beneficial to reduce the size of the domain using equi-width buckets. In that case, we assume the elements of D_i represent buckets, and N_i is the number of buckets.
- We enforce our MaxEnt distribution to be *overcomplete* [47, pp.40] (as opposed to *minimal*). More precisely, for any attribute A_i and any instance I, we have $\sum_{j \in J_i} \langle \mathbf{c}_j, \mathbf{I} \rangle = n$, which means that some statistics are redundant since they can be computed from the others and from the size of the instance n.

Note that as a consequence of overcompleteness, for any attribute A_i, one can write P as a linear expression

$$P = \sum_{j \in J_i} \alpha_j P_j \qquad (8)$$

where each P_j, $j \in J_i$ is a polynomial that does not contain the variables $(\alpha_j)_{j \in J_i}$. In Example 3.3, the 1-dimensional variables for A are α_1, α_2, and indeed, each monomial in Eq. 7 contains exactly one of these variables. One can write P as $P = \alpha_1 P_1 + \alpha_2 P_2$ where $\alpha_1 P_1$ represents the first two lines and $\alpha_2 P_2$ represents the last two lines in Eq. 7. P is also linear in β_1, β_2 and in γ_1, γ_2.

3.2 Query Answering

In this section, we show how to use the data summary to approximately answer a linear query q by returning its expected value $\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle]$. The summary $(P, \{\alpha_j\}, \Phi)$ defines a probability space on the possible worlds as it parameterizes $\Pr(I)$ (Eq. 4 and 6). We start with a well known result in the MaxEnt model. If $\mathbf{c}_\ell$ is the linear query associated with the variable α_ℓ, then

$$\mathbb{E}[\langle \mathbf{c}_\ell, \mathbf{I} \rangle] = \frac{n \alpha_\ell}{P} \frac{\partial P}{\partial \alpha_\ell}. \qquad (9)$$

We review the proof here. The expected value of $\langle \mathbf{c}_\ell, \mathbf{I} \rangle$ over the probability space (Eq. 4) is

$$\mathbb{E}[\langle \mathbf{c}_\ell, \mathbf{I} \rangle] = \frac{1}{P^n} \sum_{\mathbf{I}} \langle \mathbf{c}_\ell, \mathbf{I} \rangle \prod_j \alpha_j^{\langle \mathbf{c}_j, \mathbf{I} \rangle} = \frac{1}{P^n} \sum_{\mathbf{I}} \frac{\alpha_\ell \partial}{\partial \alpha_\ell} \prod_j \alpha_j^{\langle \mathbf{c}_j, \mathbf{I} \rangle}$$
$$= \frac{1}{P^n} \frac{\alpha_\ell \partial}{\partial \alpha_\ell} \sum_{\mathbf{I}} \prod_j \alpha_j^{\langle \mathbf{c}_j, \mathbf{I} \rangle} = \frac{1}{P^n} \frac{\alpha_\ell \partial P^n}{\partial \alpha_\ell} = \frac{n}{P} \frac{\alpha_\ell \partial P}{\partial \alpha_\ell}.$$

To compute a new linear query $\mathbf{q}$, we add it to the statistical queries c_j, associate it with a fresh variable β, and denote $P_{\mathbf{q}}$ the extended polynomial:

$$P_{\mathbf{q}}(\alpha_1, \ldots, \alpha_k, \beta) \overset{\text{def}}{=} \sum_{i=1,d} \prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{t}_i \rangle} \beta^{\langle \mathbf{q}, \mathbf{t}_i \rangle} \qquad (10)$$

Notice that $P_{\mathbf{q}}[\beta = 1] \equiv P$; therefore, the extended data summary defines the same probability space as P. With $\beta = 1$, we can apply Eq. 9 to the query $\mathbf{q}$ to derive:

$$\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle] = \frac{n}{P} \frac{\partial P_{\mathbf{q}}}{\partial \beta}. \qquad (11)$$

This leads to the following naïve strategy for computing the expected value of $\mathbf{q}$: extend P to obtain $P_{\mathbf{q}}$ and apply formula Eq. 11. One way to obtain $P_{\mathbf{q}}$ is to iterate

---

Algorithm 1 Solving for the αs

```
maxError = infinity
while maxError >= threshold do
  maxError = -1
  for each α_j do
      value = s_j(P − α_j P_{α_j})
              ───────────────────
                 (n − s_j) P_{α_j}
      α_j = value
      error = value − n α_j P_{α_j}
                      ────────────
                           P
      maxError = max(error, maxError)
```

---

over all monomials in P and add β to the monomials corresponding to tuples counted by $\mathbf{q}$. As this iteration is inefficient, Sec. 4.2 describes how to avoid modifying the polynomial altogether.

3.3 Probabilistic Model Computation

We now describe how to compute the parameters of the summary. Given the statistics $\Phi = \{(\mathbf{c}_j, s_j) : j = 1, k\}$, we need to find values of the variables $\{\alpha_j : j = 1, k\}$ such that $\mathbb{E}[\langle \mathbf{c}_j, \mathbf{I} \rangle] = s_j$ for all $j = 1, k$. As explained in Sec. 2, this is equivalent to maximizing the dual function Ψ:

$$\Psi \overset{\text{def}}{=} \sum_{j=1}^{k} s_j \ln(\alpha_j) - n \ln P. \qquad (12)$$

Indeed, maximizing P reduces to solving the equations $\partial \Psi / \partial \alpha_j = 0$ for all j. Direct calculation gives us $\partial \Psi / \partial \alpha_j = \frac{s_j}{\alpha_j} - \frac{n}{P} \frac{\partial P}{\partial \alpha_j} = 0$, which is equivalent to $s_j - \mathbb{E}[\langle \mathbf{c}_j, \mathbf{I} \rangle]$ by Eq. 9. The dual function Ψ is concave, and hence it has a single maximum value that can be obtained using convex optimization techniques such as Gradient Descent.

In particular, we achieve fastest convergence rates using a variant of Stochastic Gradient Descent (SGD) called Mirror Descent [11], where each iteration chooses some $j = 1, k$ and updates α_j by solving $\frac{n \alpha_j}{P} \frac{\partial P}{\partial \alpha_j} = s_j$ while keeping all other parameters fixed. In other words, the step of SGD is chosen to solve $\partial \Psi / \partial \alpha_j = 0$. Denoting $P_{\alpha_j} \overset{\text{def}}{=} \frac{\partial P}{\partial \alpha_j}$ and solving, we obtain:

$$\alpha_j = \frac{s_j (P - \alpha_j P_{\alpha_j})}{(n - s_j) P_{\alpha_j}}. \qquad (13)$$

Since P is linear in each α, neither $P - \alpha_j P_{\alpha_j}$ nor P_{α_j} contain any α_j variables.

We repeat this for all j, and continue this process until all differences $|s_j - \frac{n \alpha_j P_{\alpha_j}}{P}|$, $j = 1, k$, are below some threshold. Alg. 1 shows pseudocode for the solving process.

4. LOGICAL OPTIMIZATIONS

We now discuss two logical optimizations: (1) summary compression in Sec. 4.1 and (2) optimized query processing in Sec. 4.2. In Sec. 5, we discuss the implementation of these optimizations.

4.1 Compression of the Data Summary

The summary consists of the polynomial P that, by definition, has $|Tup|$ monomials where $|Tup| = \prod_{i=1}^{m} N_i$. We describe a technique that compresses the summary by factorizing the polynomial to a size closer to $O(\sum_i N_i)$ than $O(\prod_i N_i)$.

Before walking through a more complex example describing the factorization process, we show the factorized version of the polynomial from Example 3.3.

EXAMPLE 4.1. *Recall that our relation has three attributes A, B, and C with domain size of 2, and our summary has four multidimensional statistics. The factorization of P is*

$$
\begin{aligned}
P =& (\alpha_1 + \alpha_2)(\beta_1 + \beta_2)(\gamma_1 + \gamma_2) + \\
& (\gamma_1 + \gamma_2)(\alpha_1\beta_1([\alpha\beta]_{1,1} - 1) + \alpha_2\beta_2([\alpha\beta]_{2,2} - 1)) + \\
& (\alpha_1 + \alpha_2)(\beta_1\gamma_1([\beta\gamma]_{1,1} - 1) + \beta_2\gamma_1([\beta\gamma]_{2,1} - 1)) + \\
& \alpha_1\beta_1\gamma_1([\alpha\beta]_{1,1} - 1)([\beta\gamma]_{1,1} - 1) + \\
& \alpha_2\beta_2\gamma_1([\alpha\beta]_{2,2} - 1)([\beta\gamma]_{2,1} - 1).
\end{aligned}
\tag{14}
$$

As we will see, the factorization starts with a product of 1-dimensional statistics and uses the inclusion/exclusion principle to include the multi-dimensional statistics. Note that for this particular example, because the active domain is so small (eight possible tuples), the factorized polynomial is not smaller than the expanded one. We explain the polynomial size in Thm. 4.3.

We now walk through a more complex example with three attributes, A, B, and C, each with an active domain of size $N_1 = N_2 = N_3 = 1000$. Suppose first that we have only 1D statistics. Then, instead of representing P as a sum of 1000^3 monomials, i.e. $P = \sum_{i,j,k \in [1000]} \alpha_i \beta_j \gamma_k$, we factorize it to $P = (\sum \alpha_i)(\sum \beta_j)(\sum \gamma_k)$; the new representation has size $3 \cdot 1000$.

Now, suppose we add a single 3D statistic on ABC: $A = 3 \wedge B = 4 \wedge C = 5$. The new variable, call it δ, occurs in a single monomial of P, namely $\alpha_3\beta_4\gamma_5\delta$. Thus, we can compress P to $(\sum \alpha_i)(\sum \beta_j)(\sum \gamma_k) + \alpha_3\beta_4\gamma_5(\delta - 1)$.

Instead, suppose we add a single 2D range statistic on AB, say $A \in [101, 200] \wedge B \in [501, 600]$ and call its associated variable δ_1. This will affect $100 \cdot 100 \cdot 1000$ monomials. We can avoid enumerating them by noting that they, too, factorize. The polynomial compresses to $(\sum \alpha_i)(\sum \beta_j)(\sum \gamma_k) + (\sum_{i=101}^{200} \alpha_i)(\sum_{j=501}^{600} \beta_j)(\sum \gamma_k)(\delta_1 - 1)$.

Finally, suppose we have three 2D statistics and one 3D statistic: the previous one on AB plus the statistics $B \in [551, 650] \wedge C \in [801, 900]$ and $B \in [651, 700] \wedge C \in [701, 800]$ on BC and $A \in [101, 150] \wedge B \in [551, 600] \wedge C \in [801, 850]$ on ABC. Their associated variables are δ_1, δ_2, δ_3, and δ_4 (Fig. 3 shows a table of the statistics). Now we need to account for the fact that $100 \cdot 50 \cdot 100$ monomials contain both δ_1 and δ_2 and that $50 \cdot 50 \cdot 50$ monomials contain δ_1, δ_2, and δ_4. Applying the inclusion/exclusion principle, P compresses to the equation shown in Fig. 2 (the **i**, **ii**, and **iii** labels are referenced later). The size, counting only the αs, βs, and γs for simplicity, is $3000 + 1200 + 1350 + 150 + 250 + 150 + 150 + 150 = 6400 \ll 1000^3$.

Before proving the general formula for P, note that this compression is related to standard algebraic factorization techniques involving kernel extraction and rectangle coverings [25]; both techniques reduce the size of a polynomial by factoring out divisors. The standard techniques, however, are unsuitable for our use because they require enumeration of the product terms in the sum-of-product (SOP) polynomial to extract kernels and form cube matrices. Our polynomial in SOP form is too large to be materialized, making these techniques infeasible. We leave it as future work to investigate other factorization techniques geared towards massive polynomials.

$$
\begin{aligned}
P =& \overbrace{(\sum \alpha_i)(\sum \beta_j)(\sum \gamma_k)}^{(i)} + \overbrace{(\sum \gamma_k)}^{(ii)} \overbrace{(\sum_{101}^{200} \alpha_i)(\sum_{501}^{600} \beta_j)(\delta_1 - 1)}^{(iii)} \\
& + \overbrace{(\sum \alpha_i)}^{(ii)} \overbrace{\left[(\sum_{551}^{650} \beta_j)(\sum_{801}^{900} \gamma_k)(\delta_2 - 1) + (\sum_{651}^{700} \beta_j)(\sum_{701}^{800} \gamma_k)(\delta_3 - 1) \right]}^{(iii)} \\
& + \overbrace{(\sum_{101}^{150} \alpha_i)(\sum_{551}^{600} \beta_j)(\sum_{801}^{850} \gamma_k)(\delta_4 - 1)}^{(iii)} \\
& + \overbrace{(\sum_{101}^{200} \alpha_i)(\sum_{551}^{600} \beta_j)(\sum_{801}^{900} \gamma_k)(\delta_1 - 1)(\delta_2 - 1)}^{(iii)} \\
& + \overbrace{(\sum_{101}^{150} \alpha_i)(\sum_{551}^{600} \beta_j)(\sum_{801}^{850} \gamma_k)(\delta_1 - 1)(\delta_4 - 1)}^{(iii)} \\
& + \overbrace{(\sum_{101}^{150} \alpha_i)(\sum_{551}^{600} \beta_j)(\sum_{801}^{850} \gamma_k)(\delta_2 - 1)(\delta_4 - 1)}^{(iii)} \\
& + \overbrace{(\sum_{101}^{150} \alpha_i)(\sum_{551}^{600} \beta_j)(\sum_{801}^{850} \gamma_k)(\delta_1 - 1)(\delta_2 - 1)(\delta_4 - 1)}^{(iii)}.
\end{aligned}
$$

Figure 2: Example of a compressed polynomial P after applying the inclusion/exclusion principle.

We now make the following three assumptions for the rest of the paper.

- Each predicate has the form $\pi_j = \bigwedge_{i=1}^{m} \rho_{ij}$ where m is the number of attributes, and ρ_{ij} is the projection of π_j onto A_i. If $j \in J_i$ (J_i is the set of indices of the 1-dimensional statistics), then $\pi_j \equiv \rho_{ij}$. For any set of indices of multi-dimensional statistics $S \subset [k]$, we denote $\rho_{iS} \overset{\text{def}}{=} \bigwedge_{j \in S} \rho_{ij}$, and $\pi_S \overset{\text{def}}{=} \bigwedge_i \rho_{iS}$; as usual, when $S = \emptyset$, then $\rho_{i\emptyset} \equiv \texttt{true}$.
- Each ρ_{ij} is a range predicate $A_i \in [u, v]$.
- For each $\mathcal{I} \subseteq [m]$, the multi-dimensional statistics whose attributes are exactly those in $\mathcal{I}$ are disjoint; i.e., for j_1, j_2 whose attributes are $\mathcal{I}$, $\rho_{ij_1}, \rho_{ij_2} \not\equiv \texttt{true}$ for $i \in \mathcal{I}$ (i.e. there is a predicate on A_i), $\rho_{ij_1}, \rho_{ij_2} \equiv \texttt{true}$ for $i \notin \mathcal{I}$, and $\pi_{j_1} \wedge \pi_{j_2} \equiv \texttt{false}$. Attributes for different $\mathcal{I}$ may overlap, but for a particular $\mathcal{I}$, there is no overlap.

Using this, define $J_{\mathcal{I}} \subseteq [k]$ for $\mathcal{I} \subseteq [m]$ to be the set of indices of multi-dimensional statistics whose attributes are $\{A_i : i \in \mathcal{I}\}$. This means for $\mathcal{I}$ such that $|\mathcal{I}| = 1$, $J_{\mathcal{I}} = \emptyset$ because 1-dimensional statistics are not considered multidimensional statistics. Further, let $B_a = |\{\mathcal{I} : J_{\mathcal{I}} \neq \emptyset\}|$ be the number of unique multi-dimensional attributes we have statistics on and $B_s^{\mathcal{I}} = |J_{\mathcal{I}}|$ be the number of multidimensional statistics for the attribute set defined by $\mathcal{I}$. (These parameters are discussed further in Sec. 6).

Finally, define $J_{\mathcal{I}^+} \subseteq \mathcal{P}([k])^6$ for $\mathcal{I}^+ \subseteq \mathcal{P}([m])$ to be

---

[6] $\mathcal{P}([k])$ is the power set of $\{1, 2, \ldots, k\}$

the set of sets of the maximal number of multi-dimensional statistic indices from $\bigcup_{\mathcal{I} \in \mathcal{I}^+} J_{\mathcal{I}}$ such that each set's *combined* attributes are $\{A_i : i \in \bigcup \mathcal{I}^+\}$ and each set's intersection does not conflict (i.e., not `false`). In other words, for each $S \in J_{\mathcal{I}^+}$, $\rho_{iS} \notin \{\text{true}, \text{false}\}$ for $i \in \bigcup \mathcal{I}^+$ and $\rho_{iS} \equiv \text{true}$ for $i \notin \bigcup \mathcal{I}^+$.

For example, suppose we have the three 2D statistics and one 3D statistic from before: $\pi_{j_1} = A \in [101, 200] \wedge B \in [501, 600]$, $\pi_{j_2} = B \in [551, 650] \wedge C \in [801, 900]$, $\pi_{j_3} = B \in [651, 700] \wedge C \in [701, 800]$, and $\pi_{j_4} = A \in [101, 150] \wedge B \in [551, 600] \wedge C \in [801, 850]$. Then, some example $J_{\mathcal{I}^+}$ are: $J_{\{\{1,2\}\}} = \{\{j_1\}\}$, $J_{\{\{2,3\}\}} = \{\{j_2\}, \{j_3\}\}$, and $J_{\{\{1,2,3\}\}} = \{\{j_4\}\}$. $\{\{j_2, j_3\}\} \notin J_{\{\{2,3\}\}}$ because $\rho_{2j_2} \wedge \rho_{2j_3} \equiv \text{false}$. Further, $J_{\{\{1,2\},\{2,3\}\}} = \{\{j_1, j_2\}\}$ because $\rho_{2j_1} \wedge \rho_{2j_2} \not\equiv \text{false}$. However, $\{j_1, j_3\} \notin J_{\{\{1,2\},\{2,3\}\}}$ because $\rho_{2j_1} \wedge \rho_{2j_3} \equiv \text{false}$, and $\{j_4\} \notin J_{\{\{1,2\},\{2,3\}\}}$ because $j_4 \notin J_{\{1,2\}}$ and $j_4 \notin J_{\{2,3\}}$. Lastly, $J_{\{\{1,2\},\{2,3\},\{1,2,3\}\}} = \{\{j_1, j_2, j_4\}\}$

Using these definitions, we now get the compression shown in Thm. 4.2.

THEOREM 4.2. *The polynomial P is equivalent to:*

$$P = \overbrace{\left(\prod_{i \in [m]} \sum_{j \in J_i} \alpha_j \right)}^{(i)} +$$

$$\left[\sum_{\mathcal{I} \subseteq [m]} \overbrace{\left(\prod_{i \notin \mathcal{I}} \sum_{j \in J_i} \alpha_j \right)}^{(ii)} \overbrace{\sum_{\ell=1}^{B_a} \sum_{\substack{\mathcal{I}^+ \subseteq \mathcal{P}([m]), \\ |\mathcal{I}^+| = \ell, \\ \bigcup \mathcal{I}^+ = \mathcal{I}}} \sum_{S \in J_{\mathcal{I}^+}}}^{(iii)} \right.$$

$$\left. \underbrace{\left(\prod_{i \in \bigcup \mathcal{I}^+} \sum_{\substack{j \in J_i, \\ \pi_j \wedge \rho_{iS} \not\equiv false}} \alpha_j \right) \left(\prod_{j \in S} (\alpha_j - 1) \right)}_{(iii)} \right]$$

The proof uses induction on the size of $\mathcal{I}$.

To give intuition, the very first sum gives the sum over the 1D statistics, (**i**). The next sum handles the multi-dimensional statistics. When $\mathcal{I}$ is empty, (**iii**) will be zero. When there is no $\mathcal{I}^+$ matching the criteria or $J_{\mathcal{I}^+}$ is empty, that portion of the summation will be zero. When there exists some $S \in J_{\mathcal{I}^+}$, the summand sums up all 1-dimensional variables α_j, $j \in J_i$ that are in the ith projection of the predicate π_S (this is what the condition $(\pi_j \wedge \rho_{iS}) \not\equiv \text{false}$ checks) and multiplies with terms $\alpha_j - 1$ for $j \in S$.

Our algorithm to build the polynomial is non-trivial and is described in Sec. 5.1. The algorithm can be used during query answering to compute the compressed representation of $P_\mathbf{q}$ from P (Sec. 3.2) by rebuilding **iii** for the new $\mathbf{q}$. However, as this is inefficient and may increase the size of our polynomial, our system performs query answering differently, as explained in Sec. 4.2.

We now analyze the size of the compressed polynomial P. Since $B_a < 2^m$ and $\sum_{i=1}^m N_i \gg 2^m$, B_a is dominated by $\sum_{i=1}^m N_i$. For some $\mathcal{I}$, part (**ii**) of the compression is $O(\sum_{i=1}^m N_i)$. Part (**iii**) of the compression is more complex. For some $S \in J_{\mathcal{I}^+}$, the innermost summand is of size $O(\sum_{i=1}^m N_i + |S|)$. As $|S| \leq B_a \ll \sum_{i=1}^m N_i$, the summand is only $O(\sum_{i=1}^m N_i)$. This innermost summand only occurs

when $J_{\mathcal{I}^+}$ is nonempty, which happens once for all possible combinations of the B_a multi-dimensional attributes. Therefore, letting $R = \max_{\mathcal{I}^+} |J_{\mathcal{I}^+}|$ (we discuss this next), part (**iii**) is of size $O(2^{B_a} R \sum_{i=1}^m N_i)$. Putting it together, since we only are concerned with $\mathcal{I}$ such that $\bigcup \mathcal{I}^+ = \mathcal{I}$ for some $\mathcal{I}^+$ and we have 2^{B_a} relevant $\mathcal{I}^+$, the polynomial is of size $O(\sum_{i=1}^m N_i + 2^{B_a}(\sum_{i=1}^m N_i + 2^{B_a} R \sum_{i=1}^m N_i)) = O(2^{B_a} R \sum_{i=1}^m N_i)$.

Lastly, to discuss R. For a particular $\mathcal{I}^+$, $|J_{\mathcal{I}^+}|$ is the number of sets of multi-dimensional statistics whose *combined* attributes are $\{A_i : i \in \bigcup \mathcal{I}^+\}$ and whose intersection does not conflict. In the worse case, there are no conflicts (e.g. if $\bigcap \mathcal{I}^+ = \emptyset$). Then, there will be at most $\prod_{\mathcal{I} \in \mathcal{I}^+} B_s^{\mathcal{I}}$ statistics for a particular $\mathcal{I}^+$. Since the largest $\mathcal{I}^+$ has B_a elements, an upper bound on R is $\hat{B}_s^{B_a}$ where $\hat{B}_s = \max_{\mathcal{I}} B_s^{\mathcal{I}}$. We assume $\hat{B}_s \geq 2$, and therefore we get the following theorem.

THEOREM 4.3. *The size of the polynomial is $O(\hat{B}_s^{B_a} \sum_{i=1}^m N_i)$ where B_a is the number of unique multi-dimensional attribute sets and $\hat{B}_s$ is the largest number of statistics over some $\mathcal{I}$.*

In the worst case, if one gathers all possible multi-dimensional statistics, this compression will be worse than the uncompressed polynomial, which is of size $O(\prod_{i=1}^m N_i)$, approximately equal to $O((\max_i N_i)^m)$. However, in practice, $B_a < m$ and $B_s \leq \max_i N_i$ which results in a significant reduction of polynomial size to one closer to $O(\sum_{i=1}^m N_i)$ than $O(\prod_{i=1}^m N_i)$.

4.2 Optimized Query Answering

In this section, we assume that the query $\mathbf{q}$ is a counting query defined by a conjunction of predicates, one over each attribute A_i; i.e., $\mathbf{q} = |\sigma_\pi(I)|$, where

$$\pi = \rho_1 \wedge \cdots \wedge \rho_m \tag{15}$$

and ρ_i is a predicate over the attribute A_i. If $\mathbf{q}$ ignores A_i, then we simply set $\rho_i \equiv \text{true}$. Our goal is to compute $\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle]$. In Sec. 3.2, we described a direct approach that consists of constructing a new polynomial $P_\mathbf{q}$ and returning Eq. 11. However, as described in Sec. 3.2 and Sec. 4.1, this may be expensive.

We describe here an optimized approach to compute $\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle]$ directly from P. The advantage of this method is that it does not require any restructuring or rebuilding of the polynomial. Instead, it can use any optimized oracle for evaluating P on given inputs. Our optimization has two parts: a new formula $\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle]$ and a new formula for derivatives.

New formula for $\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle]$: Let π_j be the predicate associate to the jth statistical query. In other words, $\langle \mathbf{c}_j, \mathbf{I} \rangle = |\sigma_{\pi_j}(\mathbf{I})|$. The next lemma applies to any query $\mathbf{q}$ defined by some predicate π. Recall that β is the new variable associated to $\mathbf{q}$ in $P_\mathbf{q}$ (Sec. 3.2).

LEMMA 4.4. *For any ℓ variables $\alpha_{j_1}, \ldots, \alpha_{j_\ell}$ of $P_\mathbf{q}$:*
(1) If the logical implication $\pi_{j_1} \wedge \cdots \wedge \pi_{j_\ell} \Rightarrow \pi$ holds, then

$$\frac{\alpha_{j_1} \cdots \alpha_{j_\ell} \partial^\ell P_\mathbf{q}}{\partial \alpha_{j_1} \cdots \partial \alpha_{j_\ell}} = \frac{\alpha_{j_1} \cdots \alpha_{j_\ell} \beta \partial^{\ell+1} P_\mathbf{q}}{\partial \alpha_{j_1} \cdots \partial \alpha_{j_\ell} \partial \beta} \tag{16}$$

(2) If the logical equivalence $\pi_{j_1} \wedge \cdots \wedge \pi_{j_\ell} \Leftrightarrow \pi$ holds, then

$$\frac{\alpha_{j_1} \cdots \alpha_{j_\ell} \partial^\ell P_{\mathbf{q}}}{\partial \alpha_{j_1} \cdots \partial \alpha_{j_\ell}} = \frac{\beta \partial P_{\mathbf{q}}}{\partial \beta} \qquad (17)$$

PROOF. (1) The proof is immediate by noting that every monomial of $P_{\mathbf{q}}$ that contains all variables $\alpha_{j_1}, \ldots, \alpha_{j_\ell}$ also contains β; therefore, all monomials on the LHS of Eq. 16 contain β and thus remain unaffected by applying the operator $\beta \partial / \partial \beta$.

(2) From item (1), we derive Eq. 16; we prove now that the RHS of Eq. 16 equals $\frac{\beta \partial P_{\mathbf{q}}}{\partial \beta}$. We apply item (1) again to the implication $\pi \Rightarrow \pi_{j_1}$ and obtain $\frac{\beta \partial P_{\mathbf{q}}}{\partial \beta} = \frac{\beta \alpha_{j_1} \partial^2 P_{\mathbf{q}}}{\partial \beta \partial \alpha_{j_1}}$ (the role of β in Eq. 16 is now played by α_{j_1}). As P is linear, the order of partials does not matter, and this allows us to remove the operator $\alpha_{j_1} \partial / \partial \alpha_{j_1}$ from the RHS of Eq. 16. By repeating the argument for $\pi \Rightarrow \pi_{j_2}$, $\pi \Rightarrow \pi_{j_3}$, etc, we remove $\alpha_{j_2} \partial / \partial \alpha_{j_2}$, then $\alpha_{j_3} \partial / \partial \alpha_{j_3}$, etc from the RHS. $\square$

COROLLARY 4.5. *(1) Assume $\mathbf{q}$ is defined by a point predicate $\pi = (A_1 = v_1 \wedge \cdots \wedge A_\ell = v_\ell)$ for some $\ell \leq m$. For each $i = 1, \ell$, denote j_i the index of the statistic associated to the value v_i. In other words, the predicate $\pi_{j_i} \equiv (A_i = v_i)$. Then,*

$$\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle] = \frac{n}{P} \frac{\alpha_{j_1} \cdots \alpha_{j_\ell} \partial^\ell P}{\partial \alpha_{j_1} \cdots \partial \alpha_{j_\ell}} \qquad (18)$$

(2) Let $\mathbf{q}$ be the query defined by a predicate as in Eq. 15. Then,

$$\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle] = \sum_{j_1 \in J_1 : \pi_{j_1} \Rightarrow \rho_1} \cdots \sum_{j_m \in J_m : \pi_{j_m} \Rightarrow \rho_m} \frac{n}{P} \frac{\alpha_{j_1} \cdots \alpha_{j_m} \partial^m P}{\partial \alpha_{j_1} \cdots \partial \alpha_{j_m}} \qquad (19)$$

PROOF. (1) Eq. 18 follows from Eq. 11, Eq. 17, and the fact that $P_{\mathbf{q}}[\beta = 1] \equiv P$. (2) Follows from (1) by expanding $\mathbf{q}$ as a sum of point queries as in Lemma. 4.4 (1). $\square$

In order to compute a query using Eq. 19, we would have to examine all m-dimensional points that satisfy the query's predicate, convert each point into the corresponding 1D statistics, and use Eq. 18 to estimate the count of the number of tuples at this point. Clearly, this is inefficient when $\mathbf{q}$ contains any range predicate containing many point queries.

New formula for derivatives Thus, to compute $\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle]$, one has to evaluate several partial derivatives of P. Recall that P is stored in a highly compressed format, and therefore, computing the derivative may involve nontrivial manipulations. Instead, we use the fact that our polynomial is overcomplete, meaning that $P = \sum_{j \in J_i} \alpha_j P_j$, where P_j, $j \in J_i$ does not depend on any variable in $\{\alpha_j : j \in J_i\}$ (Eq. 8). Let ρ_i be any predicate on the attribute A_i. Then,

$$\sum_{j_i \in J_i : \pi_{j_i} \Rightarrow \rho_i} \frac{\alpha_{j_i} \partial P}{\partial \alpha_{j_i}} = P[\bigwedge_{j \in J_i : \pi_{j_i} \not\Rightarrow \rho_i} \alpha_j = 0] \qquad (20)$$

Thus, in order to compute the summation on the left, it suffices to compute P after setting to 0 the values of all variables α_j, $j \in J_i$ that do not satisfy the predicate ρ_i (this is what the condition $\pi_{j_i} \not\Rightarrow \rho_i$ checks).

Finally, we combine this with Eq. 19 and obtain the following, much simplified formula for answering a query $\mathbf{q}$, defined by a predicate of the form Eq. 15:

$$\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle] = \frac{n}{P} P[\bigwedge_{i=1, m} \bigwedge_{j \in J_i : \pi_{j_i} \not\Rightarrow \rho_i} \alpha_j = 0] \qquad (21)$$

In other words, we set to 0 all 1D variables α_j that correspond to values that do *not* satisfy the query, evaluate the polynomial P, and multiply it by $\frac{n}{P}$ (which is a precomputed constant independent of the query). For example, if the query ignores an attribute A_i, then we leave the 1D variables for that attribute, α_j, $j \in J_i$, unchanged. If the query checks a range predicate, $A_i \in [u, v]$, then we set $\alpha_j = 0$ for all 1D variables α_j corresponding to values of A_i outside that range.

EXAMPLE 4.6. *Consider three attributes A, B, and C each with domain 1000 and three multi-dimensional statistics: one AB statistic $A \in [101, 200] \wedge B \in [501, 600]$, two BC statistics $B \in [551, 650] \wedge C \in [801, 900]$ and $B \in [651, 700] \wedge C \in [701, 800]$, and one ABC statistic $A \in [101, 150] \wedge B \in [551, 600] \wedge C \in [801, 850]$. The polynomial P is shown in Fig. 2. Consider the query $\mathbf{q}$:*

```
SELECT COUNT(*) FROM R
WHERE A in [36,150] AND C in [660,834]
```

We estimate $\mathbf{q}$ using our formula $\frac{n}{P} P[\alpha_{1:35} = 0, \alpha_{151:1000} = 0, \gamma_{1:659} = 0, \gamma_{835:1000} = 0]$. There is no need to compute a representation of a new polynomial.

5. SYSTEM OPTIMIZATIONS

In Sec. 4, we discussed two main optimizations: polynomial compression and query answering by setting certain 1D variables to zero. We now discuss how to implement these optimizations efficiently and analyze the runtime.

5.1 Building the Polynomial

Recall from Thm. 4.2 that building our compressed polynomial starts with the product of the sum of all 1D statistics and then uses the inclusion/exclusion principle to modify the terms to include the correct multi-dimensional statistics. The terms that need to be modified are those that satisfy the predicates associated with the sets of non-conflicting multi-dimensional statistics. i.e., for some $\mathcal{I}^+$, the terms to be modified are the 1D terms α_j such that $\pi_j \wedge \pi_S \not\equiv \texttt{false}$ for $S \in J_{\mathcal{I}^+}$. The algorithmic challenge is how we find non-conflicting statistics $J_{\mathcal{I}^+}$ for some $\mathcal{I}^+$ and, once we know $J_{\mathcal{I}^+}$, how we find the 1D statistics that need to be modified.

To solve the latter problem, assume we have some $J_{\mathcal{I}^+}$. Since each multi-dimensional statistic is a range predicate over the elements in our domain and we have complete 1D statistics over the elements in our domain, once we know the range predicate, π_S for $S \in J_{\mathcal{I}^+}$, we can easily find the associated 1D statistics.

Take the example in Fig. 2 which we will refer to throughout this section. If we know that $J_{\{\{1,2\}, \{2,3\}, \{1,2,3\}\}} = \{\{j_1, j_2, j_4\}\}$, then, by examining the range predicates associated with those three multi-dimensional statistics, we can determine that α_i for $i \in [101, 150]$, β_j for $j \in [551, 600]$, and γ_k for $k \in [801, 850]$ are the 1D statistics that need to be modified to include δ_1, δ_2, and δ_4.

The other problem is how we find the groups of multi-dimensional statistics that do not conflict for some group

Algorithm 2 Unoptimized Building P

```
// add part i to P
P = 1DProdSum(1DStats) ── O(mN)
for (k in [1:B_a]) do
    for (idx in combinations(k, B_a)) do
        // add part ii to P
        P.addTerms(1DProdSum(1DStats[not idx]))
        satGrps = findNoConflictGrps(multiDStats[idx])        ── O((k-1)(mN)²B̂_s^k)
        // add part iii to P
        for (group in satGrps) do                             } O(mN + B̂_s^k)
            P.addTerms(buildTerm(group))
```

of attribute sets (i.e. the $J_{\mathcal{I}+} \neq \emptyset$). To solve this, we assume we are given four inputs: a list of attributes, a list of B_a multi-dimensional attribute sets (e.g. $[AB, BC, ABC]$ for Fig. 2), a dictionary of the 1D statistics with attributes A_i as keys (denoted `1DStats`), and a dictionary of multi-dimensional statistics with indices into the list of B_a attributes sets as keys (denoted `multiDStats`).

A straightforward algorithm to build the polynomial is shown in Alg. 2 where the red notations indicate the time complexity of each section of pseudo code. The function `combinations`(k, B_a) generates a list of all possible length k index sets from $[1, B_a]$. Note that we abuse the notation for dictionary selection slightly in that if `idx` is $\{1, 2\}$, `multiDStats[idx]` would select both the multi-dimensional stats of 1 and 2, e.g. AB and BC, and `1DStats[not idx]` would select no 1D stats since all attributes are used in AB and BC. The function `buildTerm`(group) builds the term shown in the last line of Thm. 4.2. It generates a sum of the 1D statistics associated with the group and multiplies the sum by one minus the multi-dimensional variables in the group.

The last function to discuss is findNoConflictGrps which returns a dictionary with keys as sets of multi-dimensional attribute indices and values of groups of conflict free statistics. For example, for $k = 2$, a key would be $\{1, 2\}$ with value $\{\delta_1, \delta_2\}$ indicating that δ_1 and δ_2 do not conflict. The algorithm works by treating each multi-dimensional index set in `idx` as a relation with rows of the statistics associated with that index set. It then computes a theta-join over these relations with the join condition being if the statistics are conflict free.

For example, δ_1 and δ_2 are conflict free but not δ_1 and δ_3. Further, statistics over disjoint attributes sets are also conflict free. If we had a relation $R(A, B, C, D)$ and some statistic over AB and another over CD, all of those multi-dimensional statistics from AB would be satisfiable with all other from CD.

To understand the runtime complexity of the algorithm, start with the function buildTerm(group). For ease of notation, we will denote $N = \max_i N_i$. Recall that mN is the total number of distinct values across all attributes, B_a is the number of attribute sets, and $\hat{B}_s$ is the largest number of statistics per attribute set. The runtime of buildTerm(group) for a single satGrps of size k is $O(mN + \hat{B}_s^k)$ because a single satGrps will only add each 1D statistic at most once and at most $\hat{B}_s^k$ correction terms. This runtime also includes the time for 1DProdSum because the 1D statistics that are not in satGrps will be added in 1DProdSum.

The runtime of findNoConflictGrps involves computing the cross product of the multi-dimensional statistics and

comparing the 1D statistics associated with each multi-dimensional statistic to determine if there is a conflict. Specifically, it computes a right deep join tree of the multi-dimensional statistics, and at each step in the tree, iterates over the 1D statistics in each right child conflict free group to see if there is a conflict or not with one of the incoming left child multi-dimensional statistics.

Fig. 3 shows the findNoConflictGrps join algorithm for the attribute sets AB, BC, and ABC with one added statistic δ_5 on BC of $B \in [401, 550] \wedge C \in [751, 850]$. The function to find and return a single conflict free group is CFG(δ_L, $\{\delta\}_S$) (stands for conflict free group) where δ_L stands for the left multi-dimensional statistic and $\{\delta\}_S$ stands for the right, current conflict free group. The S subscript is because we are building a new set of multi-dimensional statistics to add to some $J_{\mathcal{I}+}$. We are abusing notation slightly because in Sec. 4, S stood for the set of indices whereas here, it sands for the set of statistic variables.
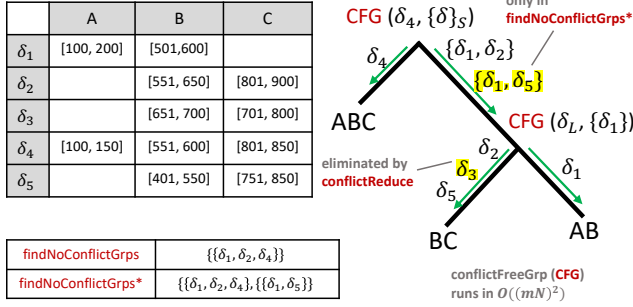
CFG first determines which attributes are shared by δ_L and $\{\delta\}_S$. Then, for each such attribute, it iterates over δ_L's associated 1D statistics (at most mN of them) and checks if at least one of these statistics also exists in the 1D statistics associated with $\{\delta\}_S$ (containment has runtime mN). This ensures, for all attributes A_i, that $\rho_{iL} \wedge \rho_{iS} \not\equiv$ `false`. If A_i is shared, then $\rho_{iL} \wedge \rho_{iS}$ will contain the shared 1D statistic found earlier, and if A_i is not shared, then $\rho_{iL} \wedge \rho_{iS} \equiv$ `true`. Note that a conflict free group is not always found. Therefore, the runtime of the join is $O(2(mN)^2 \hat{B}_s^3)$. For joins of arbitrary size, the runtime is $O((mN)^2(k-1)\hat{B}_s^k)$.

Putting it all together we get the runtime of

$$
= mN + \sum_{k=1}^{B_a} \binom{B_a}{k} [mN + (k-1)(mN)^2 \hat{B}_s^k + \hat{B}_s^k]
$$

$$
= mN + (2^{B_a} - 1)mN + (\hat{B}_s + 1)^{B_a} - 1 +
$$
$$
(mN)^2 \left[\left[\sum_{k=0}^{B_a} \binom{B_a}{k} [k\hat{B}_s^k - \hat{B}_s^k] \right] + 1 \right]
$$

$$
= mN + (2^{B_a} - 1)mN + (\hat{B}_s + 1)^{B_a} - 1 + (mN)^2 +
$$
$$
(mN)^2 \left[\sum_{k=0}^{B_a} \binom{B_a}{k} k\hat{B}_s^k \right] - (mN)^2 (\hat{B}_s + 1)^{B_a}
$$

$$
= mN + (2^{B_a} - 1)mN + (\hat{B}_s + 1)^{B_a} - 1 + (mN)^2 +
$$
$$
(mN)^2 B_a \hat{B}_s (\hat{B}_s + 1)^{B_a - 1} - (mN)^2 (\hat{B}_s + 1)^{B_a}
$$

This algorithm, however, is suboptimal because it must run findNoConflictGrps for all 2^{B_a} attribute sets. It is better to run findNoConflictGrps once for the all multi-dimensional statistics (i.e. compute the full theta-join of all B_a attribute sets) and reconstruct the smaller groups without paying the cost of checking for conflicts (i.e. perform selections over the full theta-join). Further, there are some statistics that will not appear in any other term besides when $k = 1$ in the loop above. Take δ_3, for example. It is handled in line 2 of Fig. 2 but does not appear later on. These insights lead to a more optimized algorithm in Alg. 3.

The function conflictReduce is like a semi-join reduction. It removes multi-dimensional statistics that do not appear in any conflict free group later on. For example, redMDStats would only contain δ_1, δ_2, and δ_4. The function findNoConflictGrps* acts just as findNoConflictGrps except instead of computing an inner theta join, it computes a full outer

	A	B	C
δ_1	[100, 200]	[501, 600]	
δ_2		[551, 650]	[801, 900]
δ_3		[651, 700]	[701, 800]
δ_4	[100, 150]	[551, 600]	[801, 850]
δ_5		[401, 550]	[751, 850]

findNoConflictGrps	$\{\{\delta_1, \delta_2, \delta_4\}\}$
findNoConflictGrps*	$\{\{\delta_1, \delta_2, \delta_4\}, \{\{\delta_1, \delta_5\}\}$

Figure 3: Figure of findNoConflictGrps and findNoConflictGrps* join algorithm for the three attribute sets AB, BC, and ABC. The highlighted statistics of δ_3 and $\{\delta_1, \delta_5\}$ are the difference between the unoptimized and optimized algorithms.

Algorithm 3 Optimized Building P

```
// add part i to P
P = 1DProdSum(1DStats)                              — O(mN)
// add terms when k = 1
for (idx in [1:B_a]) do
  // add part ii to P
  P.addTerms(1DProdSum(1DStats[not idx]))    ⎫
  // add part iii to P                        ⎬  O(mNB_a +
  for (group in multiDStats[idx]) do          ⎪    B_a B̂_s)
    P.addTerms(buildTerm(group))             ⎭
redMDStats = conflictReduce(multiDStats)      — O((B_a choose 2)(B_a B̂_s)^2)
satGrps = findNoConflictGrps*(redMDStats)
                                O((B_a − 1)(mN)^2 B̂_s^{B_a})*
for (k in [2:B_a]) do
  for (idx in combinations(k, B_a)) do
    // add part ii to P
    P.addTerms(1DProdSum(1DStats[not idx]))
    // add part iii to P
    for (group in satGrps) do
      P.addTerms(buildTerm(group[idx]))
```
$$O((2^{B_a} - B_a - 1) * (mN + \hat{B}_s^{B_a}))$$

---

theta join. The reason being that satGrps needs to keep track of all conflict free groups even if they contain less that B_a statistics. For example, take the δ_5 statistic of BC as shown in Fig. 3. δ_1 and δ_5 are conflict free, but δ_1, δ_5, and δ_4 are not conflict free because δ_5 conflicts with δ_4 in attribute B. In this case, findNoConflictGrps* would return a dictionary with the keys $\{1, 2, 3\}$ and $\{1, 2\}$ and values $\{\delta_1, \delta_2, \delta_4\}$ and $\{\delta_1, \delta_5\}$, respectively. The outer join ensures $\{\delta_1, \delta_5\}$ is not lost. Note the time complexity of findNoConflictGrps and findNoConflictGrps* is the same because they must compute the full cross product and then filter. Lastly, the group[idx] index selection selects the statistics associated with the attribute sets in idx.

We will now show that this algorithm's time complexity is more optimal than Alg. 2 because although it loops through satGrps, selecting out a subterm is faster than rebuilding one, especially after semi-join reduction. Note that if group[idx] is has already been added to the polynomial from a previous group, it is just ignored when addTerms is called. As the red notation indicates, the runtime of the first for loop is $O(mNB_a + B_a\hat{B}_s)$ because for each idx, there are $\hat{B}_s$ multi-dimensional statistics and mN 1D statistics to add the term.

The runtime of conflictReduce involves comparing pairs of multi-dimensional statistics to see if they will participate in any conflict free groups of size two or more. For
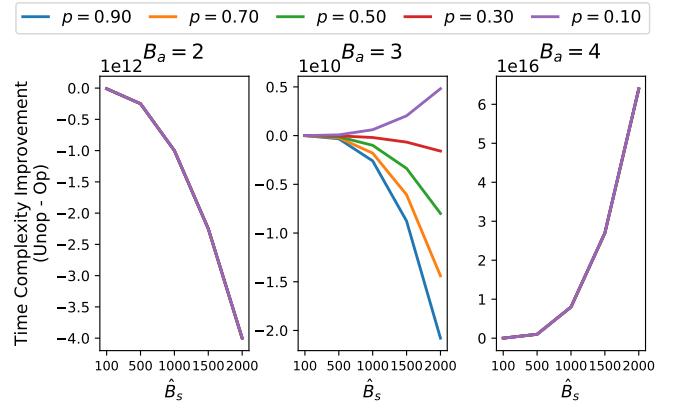
Figure 4: Algorithmic complexity improvement of the unoptimized algorithm over the optimized algorithm (complexity difference) for $mN = 5000$ and varying B_a, $\hat{B}_s$, and p. A positive number indicates the optimized algorithm is faster.

each $\binom{B_a}{2}\hat{B}_s^2$ possible pairs of multi-dimensional statistics, the conflict checking requires examining the 1D statistics of the pair, just like in $CFG(\delta_L, \{\delta\}_S)$. The next function, findNoConflictGrps*, has the same runtime as before except instead of being run for all k, it is run only once for $k = B_a$.

The last part to analyze is the for loop that iterates over all satGrps. In our runtime analysis, we add a percentage $p \in [0, 1]$ to indicate that only a fraction of the possible $\hat{B}_s^{B_a}$ groups are used in the last loop. This decrease is because of conflictReduce and because, in practice, there are drastically fewer than $\hat{B}_s^{B_a}$ resulting conflict free groups. In practice, $p \leq 0.1$. As the inner most for loop is that same as in Alg. 2 except for $k = B_a$, the runtime is $O(\hat{B}_s^{B_a} + mN)$. As this happens for all combinations from $k = 2, B_a$, the overall runtime is $O((2^{B_a} - B_a - 1)(p\hat{B}_s^{B_a} + mN))$ where the minus is because k starts at two instead of zero.

Adding up the different runtime components, we get the overall runtime of Alg. 3 is

$$= mN + B_a(mN + \hat{B}_s) + \binom{B_a}{2}(mN\hat{B}_s)^2 +$$
$$(B_a - 1)(mN)^2\hat{B}_s^{B_a} + (2^{B_a} - B_a - 1)(mN + p\hat{B}_s^{B_a})$$

To show the improvement of the optimized algorithm, Fig. 4 shows the runtime difference between Alg. 2 and Alg. 3 (i.e. Alg. 2 - Alg. 3) when $mN = 5000$ (the trends are similar for other values of mN). The three columns are for $B_a = 2, 3, 4$, the colors represents the different values of p, and $\hat{B}_s$ varies from 100 to 2000. Note that the y-axis of the three columns are on a different scale in order to show the variation between the different values of p.

We see that p matters for $B_a = 3$, and when p falls between 0.3 and 0.1, the optimized version is faster. As, in practice, $p \leq 0.1$, the optimized version is best for $B_a > 2$. The trend shown in $B_a = 4$ is the same for $B_a > 4$ and thus not included in the plot. This shows that asymptotically, Alg. 3 is optimal.

5.2 Polynomial Evaluation

Recall from Sec. 4.2 and Example 4.6 that for a linear query $\mathbf{q}$ defined by some predicate π, we can answer the query in expectation (i.e. $\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle]$) by setting all 1D variables

α_j that correspond to values that do *not* satisfy π to zero. This is more efficient than taking multiple derivatives, but simply looping over all variables can still be too slow as the compressed polynomial can, at worst, have exponentially many variables (see Thm. 4.2).

To improve performance, we implement four main optimizations: (1) storing the compressed polynomial in memory, (2) parallelizing the computation, (3) fast containment check using bit vectors, (4) caching of subexpression evaluation. The first and second, storing in memory and parallelization, are straightforward, standard techniques that improve looping computations. Note, we can parallelize the computation because each polynomial term can be evaluated independently.

The next optimization, using bit vectors, is to optimize both findNoConflictGrps and determining if a variable needs to be set to zero or not during query evaluation. It is important to understand that the polynomial is hierarchical with nested levels of sums of products of sums. For each subterm (i.e. sum or product term in our polynomial), we store (a) a map with variable keys and values of the nested subterms containing that variable, (b) a bit vector of which attributes are contained in the subterm, and (c) a bit vector of which multi-dimensional attribute sets are contained in the subterm.

Take Example 4.6 which references Fig. 2. Take the subterm referenced by **i**. This subterm has a map of all variables pointing to one of three nested sum subterms. The attribute bit vector has a 1 in all places, representing it contains all possible attributes, and the multi-dimensional bit vector is all 0s. Now take the subterm $(\sum_{551}^{650} \beta_j)(\sum_{801}^{900} \gamma_k)(\delta_2 - 1)$. It has a map of only β variables from $[551, 650]$, γ variables from $[801, 900]$, and δ_2 all pointing to one of three nested subterms. The attribute bit vector would only have 1 in the B and C dimensions, and the multi-dimensional bit vector would have a 1 in the dimension representing the attribute set BC.

These objects allow us to quickly check if some 1-dimensional or multi-dimensional statistic is contained in the term or if there are any variables that need to be set to zero (by using the attribute bit vectors) and which subterms those variables are in. To further see the benefit of this optimization, recall the runtime analysis from Sec. 5.1 where findNoConflictGrps required $(mN)^2$ steps to check if two statistics were conflict free as it iterates over all 1D statistics. Using maps with variable keys allows us to quickly check if a 1D statistic is contained in another, bringing the runtime down to mN for a single pair. The attribute bit vectors can also allow us to skip iterating over subsets of 1D statistics by quickly checking which attributes two statistics share. If some attribute is not shared between two statistics, then that attribute can cause no conflicts and does not need to be iterated over.

This leads to the last technique of caching. Caching is used to avoid recomputing subterms and takes advantage of the attribute bit vectors and variable hash maps described above. Since we solve for all the variables α_j of our model once and they remained fixed throughout query answering, if there is a subterm of our model that does not contain any variable the needs to be set to zero, we can reuse that subterm's value. We store this value along with the map and bit vectors.

By utilizing these techniques, we reduced the time to learn the model (solver runtime) from 3 months to 1 day and saw a decrease in query answering runtime from around 10 sec to 500 ms (95% decrease). More runtime results are in Sec. 7.

6. STATISTIC SELECTION

In this section, we discuss how we choose the multi-dimensional statistics. We investigate different heuristic techniques for both finding optimal statistic ranges and re-ordering the data prior to statistic collection optimally. Recall that our summary always includes all 1D statistics of the form $A_i = v$ for all attributes A_i and all values v in the active domain D_i. We describe here how to tradeoff the size of the summary for the precision of the MaxEnt model.

6.1 Optimal Ranges

The first choice we make is to include only 2D statistics. It has been shown that restricting to pairwise correlations offers a reasonable compromise between the number of statistics needed and the summary's accuracy [46]. This means each multi-dimensional statistic predicate π_j is equivalent to a range predicate over two attributes $A_{i_1} \in [u_1, v_1] \land A_{i_2} \in [u_2, v_2]$. If A_{i_1} and A_{i_2} are two dimensions of a rectangle, π_j defines a sub-rectangle in this space. As the 2D predicates are disjoint, if π_{j_1} and π_{j_2} both define rectangles over A_{i_1} and A_{i_2}, then these rectangles do not overlap.

As mentioned in Sec. 4.1, we have two parameters to consider: B_a, the number of distinct attribute pairs we gather statistics on, and B_s, the number of statistics to gather per each attribute pair. We choose to make B_s be the same for all multi-dimensional statistics. The problem is as follows: given B_a and B_s, which B_a attribute pairs $A_{i_1} A_{i_2}$ do we collect statistics on and which B_s statistics do we collect for each attribute pair? This is a complex problem, and we make the simplifying assumption that B_s, the number of statistics, is given, but we explore different choices of B_a in Sec. 7. We leave it to future work to investigate automatic techniques for determining the total budget, $B_a * B_s$.

Given B_a, we consider two different approaches when picking pairs: attribute correlation and attribute cover. The first focuses only on correlation by picking the set of attribute pairs that have the highest combined correlation[7] such that every pair has at least one attribute not included in any previously chosen, more correlated pair. This is similar to computing a Chow-Liu tree which is a maximum weight spanning tree over a graph where attributes are nodes and edge weights are the mutual information between pairs of attributes [15]. The difference is that we use the chi-squared metric rather than the mutual information as chi-squared is a common independence test for categorical data. We leave it to future work to evaluate different correlation metrics.

The second approach focuses on attribute cover by picking the set of pairs that cover the most attributes with the highest combined correlation. For example, if $B_a = 2$ and we have the attribute pairs BC, AB, CD, and AD in order of most to least correlated, if we only consider correlation, we would choose AB and BC. However, if we consider attribute cover, we would choose AB and CD. We experiment with both of these choices in Sec. 7.

---

[7]This can be found by calculating, for all attribute pairs, the chi-squared value on the contingency table of A_{i_1} and A_{i_2} and sorting from highest to lowest chi-squared value.

Next, we assume for each attribute A_i, its domain D_i is ordered and viewed as an array such that $D_i[1] \leq D_i[2] \leq \ldots$. This allows us to define a $D_{i_1} \times D_{i_2}$ space (a $N_{i_1} \times N_{i_2}$ matrix denoted $\mathcal{M}$) representing the frequency of attribute pairs. In particular, for some $x \in [1, N_{i_1}]$ and $y \in [1, N_{i_2}]$, $\mathcal{M}[x, y] = |\sigma_{A_{i_1} = D_{i_1}[x] \wedge A_{i_2} = D_{i_2}[y]}(I)|$. Our goal is to choose the best B_s 2D range predicates $[l^x, u^x] \times [l^y, u^y]$ where l^x and u^x are lower and upper index bounds on the x axis (likewise for the y axis). We consider three heuristics and show experimental results to determine which technique yields, on average, the lowest error on query results.

LARGE SINGLE CELL In this heuristic, the range predicates are single point predicates, $A_{i_1} = D_{i_1}[x] \wedge A_{i_2} = D_{i_2}[y]$, and we choose the points (x, y) as the B_s most popular values in the two dimensional space; i.e., the B_s largest values of $|\sigma_{A_{i_1} = D_{i_1}[x] \wedge A_{i_2} = D_{i_2}[y]}(I)|$.

ZERO SINGLE CELL In this heuristic, we select the empty/zero/nonexistent cells; i.e., we choose B_s points (x, y) s.t. $\sigma_{A_{i_1} = D_{i_1}[x] \wedge A_{i_2} = D_{i_2}[y]}(I) = \emptyset$. If there are fewer than B_s such points, we choose the remaining points as in LARGE SINGLE CELL. The justification for this heuristic is that, given only the 1D statistics, the MaxEnt model will produce false positives ("phantom" tuples) in empty cells; this is the opposite problem encountered by sampling techniques, which return false negatives. This heuristic has another advantage because the value of α_j in P is always 0 and does not need to be updated during solving.

COMPOSITE This method partitions $\mathcal{M}$ into a set of B_s disjoint rectangles and associates one statistic with each rectangle. For example if π_{j_1} is $A_{i_1} \in [u_1, v_1] \wedge A_{i_2} \in [u_2, v_2]$ and π_{j_2} is $A_{i_1} \in [u_3, v_3] \wedge A_{i_2} \in [u_4, v_4]$, then the composite statistic of π_{j_1} and π_{j_2} is $A_{i_1} \in ([u_1, v_1] \vee [u_3, v_3]) \wedge A_{i_2} \in ([u_2, v_2] \vee [u_4, v_4])$. We choose to combine the statistics by an attribute-wise union because our factorization algorithm requires it. Part (**iii**) of Thm. 4.2 multiplies the multi-dimensional statistic correction term (i.e. $(\delta - 1)$) by a sum of the 1D statistics associated with it. In our example, we would multiply the composite statistic correction term by $(\alpha_{u_1} + \ldots + \alpha_{v_1} + \alpha_{u_3} + \ldots + \alpha_{v_3})(\alpha_{u_2} + \ldots + \alpha_{v_2} + \alpha_{u_4} + \ldots + \alpha_{v_4})$, which can be represented by a rectangle or bounding box. As we must maintain that the composite statistics can be represented by disjoint rectangles, we use an adaptation of K-D trees to partition the data.

Recall that a K-D tree partitions a k-dimensional space by iterating over each axis i and splitting the space at the median of the ith axis. Each child is then partitioned on the $i + 1$ axis. The only difference between our K-D tree algorithm and the traditional one is our splitting condition. Instead of splitting on the median, we split on the value that has the lowest sum squared average value difference.

For a child partition with boundary $[l^x, u^x] \times [l^y, u^y]$, the split condition for the x axis is shown in Eq. 22 where $\bar{s}_l$ is the average value of the left partition candidate; i.e.,

$$\bar{s}_l = \frac{\sum_{(x,y) \in [l^x, m^x] \times [l^y, u^y]} (\mathcal{M}[x, y])}{(m^x - l^x + 1)(u^y - l^y + 1)}.$$

$\bar{s}_r$ is for the right partition candidate which uses $[m^x + 1, u^x]$ instead of $[l^x, m^x]$.
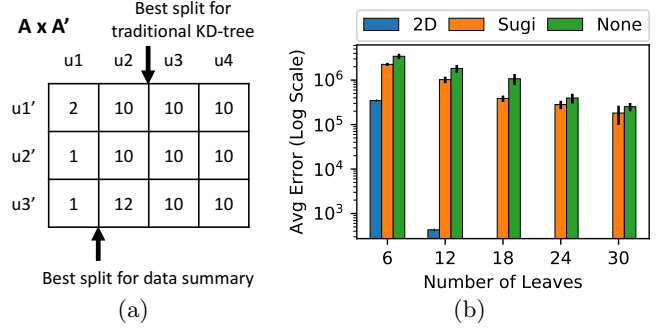

Figure 5: (a) Example K-D tree showing the traditional split on the median versus our split minimizing average error. (b) Comparison of not sorting, using SUGI sort, or using 2D sort before running the K-D tree algorithm.

$$\arg\min_{m^x} \left[\sum_{(x,y) \in [l^x, m^x] \times [l^y, u^y]} (\mathcal{M}[x, y] - \bar{s}_l)^2 \right.$$
$$\left. + \sum_{(x,y) \in [m^x+1, u^x] \times [l^y, u^y]} (\mathcal{M}[x, y] - \bar{s}_r)^2 \right]^{1/2}. \quad (22)$$

An equivalent expression is used for the y axis.

We choose this split because we want our K-D tree to best represent the true values. Suppose we have cell counts on dimensions A and A' as shown in Fig. 5a. For the next vertical split, if we followed the standard K-D tree algorithm, we would choose the second split. Instead, our method chooses the first split. Using the first split minimizes the sum squared error.

Our **COMPOSITE** method repeatedly splits the attribute domains D_{i_1} and D_{i_2} (alternating) by choosing the split value following Eq. 22 until it exhausts the budget B_s. Then, for each rectangle $[l_j^x, u_j^x] \times [l_j^y, u_j^y]$ in the resulting K-D tree, it creates a 2D statistic $(\mathbf{c}_j, s_j)$, where the query $\mathbf{c}_j$ is associated with the number of tuples satisfying the 2D range predicate and the numerical value

$$s_j \stackrel{\text{def}}{=} |\sigma_{A_{i_1} \in [D_{i_1}[l_j^x], D_{i_1}[u_j^x]] \wedge A_{i_2} \in [D_{i_2}[l_j^y], D_{i_2}[u_j^y]]}(I)|.$$

In Sec. 7.5.1 we evaluate the three different heuristic selection techniques.

6.2 Optimal Ordering

Here we describe how to improve the **COMPOSITE** method by reordering the domains of the attributes, i.e. the values in the matrix $\mathcal{M}$, because the split condition (Eq. 22) depends on the similarity of values within the bounds $[l^x, u^x] \times [l^y, u^y]$. Since our K-D tree relies on the sort order of the underlying matrix, we can permute the rows and columns before building the K-D tree to achieve a lower error.

To show how data ordering can improve the average sum squared error across the leaves, take the K-D tree plots in Fig. 6. The K-D tree splits are shown in black lines on top of frequency heatmaps. The average error is printed below the x-axis. The trees are built on 12 by 12 data with individual cell frequencies ranging from 0 to 4,000,000. The

No Sort **2D Sort**
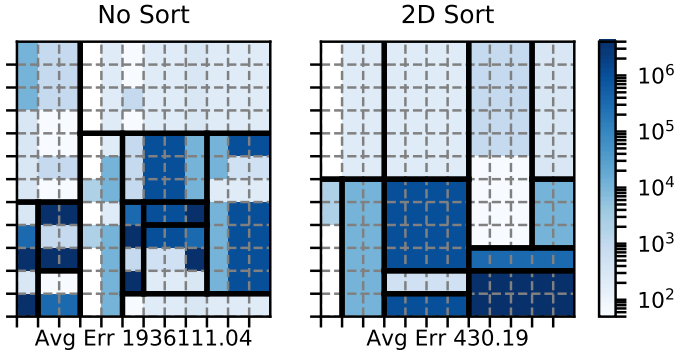
Avg Err 1936111.04   Avg Err 430.19

Figure 6: Plots showing the frequency heatmaps and the K-D trees built on data that is unsorted (left) and sorted using the 2D sort algorithm (right). The average K-D tree leaf error is shown below.

Figure 7: Sorting a matrix's rows by 2D sort (top) and SUGI sort (bottom).

data is constructed such that there is an optimal ordering that achieves 0 average sum squared error. The left plot is unordered while the right plot more optimally sorts the data (we describe the sorting in Sec. 6.3). It can be seen that (b) has grouped together similar values which means leaves have lower error. To formalize the problem, let the matrix $\mathcal{M} = D_{i_1} \times D_{i_2}$ of size $N_{i_1} \times N_{i_2}$ be the frequency of values in the domains of attributes A_{i_1} and A_{i_2}. For some index point (x, y), $\mathcal{M}[x, y] = |\sigma_{A_{i_1} = D_{i_1}[x] \wedge A_{i_2} = D_{i_2}[y]}(I)|$. Denote the set of K-D tree leaves generated from running the K-D tree algorithm as $\mathrm{KD}(\mathcal{M}) = \{[l_j^x, u_j^x] \times [l_j^y, u_j^y] : j = 1, B_s\}$. The K-D tree error is

$$\mathrm{err}(\mathrm{KD}(\mathcal{M})) \stackrel{\mathrm{def}}{=} \frac{1}{B_s} \left[\sum_{(x,y) \in [l_j^x, u_j^x] \times [l_j^y, u_j^y]} (\mathcal{M}[x, y] - \bar{s}_j)^2 \right]^{1/2} \tag{23}$$

where $\bar{s}_j$ is the average value per cell; i.e., $\bar{s}_j = s_j/(u_j^x - l_j^x + 1)(u_j^y - l_j^y + 1)$.

Our goal is to solve

$$\underset{\pi_x, \pi_y}{\arg \min} \, \mathrm{err}(\mathrm{KD}(\pi_x \mathcal{M} \pi_y)) \tag{24}$$

where π_x and π_y are row and column permutation matrices, respectively.

To solve this, we rely on heuristic techniques.

6.3 Heuristic Sorts

Inspired by the work in finding optimal matrix reorderings for data visualization and Rectangle Rule List minimization [34, 8, 5], we experiment with two different heuristic sort algorithms described in [34] to more optimally order $\mathcal{M}$ and reduce Eq. 23. At a high level, these heuristic techniques aim to permute a matrix to group together similar values. In doing so, this helps to mimimize our K-D tree error because a rectangle around these values will have lower error.

Both of the sort heuristic algorithms alternate between reordering the rows and columns until either a maximum iteration has been reached or there is no change to the sort order. The first sort algorithm, Sugiyama sort (**SUGI**), is traditionally used on binary data and sorts the rows (columns) by the average index of the one-valued columns (rows). We modify the sort to sort by the average index of zero-valued
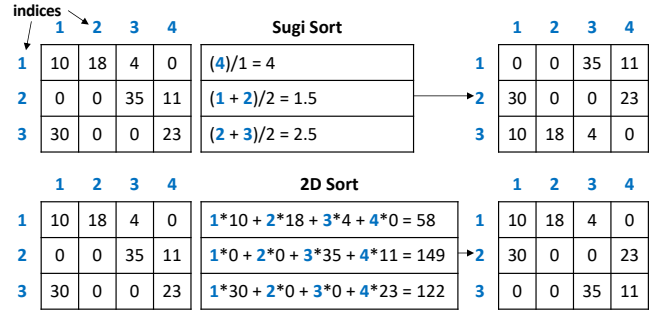
columns (rows) instead to encourage more zero-valued rectangles and lower the likelihood of having a zero-valued cell in a non-zero rectangle. The second sort, **2D sort**, sorts the rows (columns) by the sum of index times the values in the columns (rows); i.e., a weighted column (row) sum weighted by the index value. Note that the index starts at one, not zero.

For example, Fig. 7 shows a matrix with index values in blue next to the rows and columns. The top diagram shows how **SUGI** sort reorders the rows of the matrix by the average index value of the zeros. As the rows are sorted in ascending order, the middle row moves to the top, the third row to the middle, and the first row to the bottom.

The second diagram shows how **2D** sort reorders the rows of the matrix by the index weighted sum of the values. In this case, the result is that the second and third rows switch. The sorts would then continue by reordering the columns by the same techniques and so on.

To evaluate the two different sort heuristics, we first generate a 12 x 12 matrix $\mathcal{M}$ that has an optimal permutation order such that $\mathrm{err}(\mathrm{KD}(\mathcal{M})) = 0$ when using 12 K-D tree leaves. We randomly permute the rows and columns of $\mathcal{M}$ ten times and compare running the K-D tree algorithm directly on the unsorted matrix versus first doing **SUGI** or **2D** sort with various values for the number of K-D tree leaves.

Fig. 5b shows the difference in average K-D tree error (Eq. 23) and standard deviation for the three methods across ten trials with the number of leaves varying from 6 to 30. You can see that **2D** sort greatly outperforms **SUGI** sort and has no standard deviation because it always reaches the same sort order. It also very quickly converges to having zero error, but it does not learn the optimal order because it does not get zero error with 12 leaves. **2D** sort's success is due to the fact that it takes values into account as well as the index position, therefore grouping together cells with similar frequencies. **SUGI** sort, on the other hand, merely tries to group together the zeros. We do see, however, that **SUGI** sort is better than no sort.

We show in Sec. 7.5.2 how using **2D** sort impacts the overall query error of our MaxEnt technique.

7. EVALUATION

In this section, we evaluate the performance of EntropyDB in terms of query accuracy and query execution time. We compare our approach to uniform sampling and stratified sampling.

7.1 Implementation

We implemented our polynomial solver and query evaluator in Java 1.8, in a prototype system that we call EntropyDB. We created our own polynomial class and variable types to implement our factorization. We parallelized our polynomial evaluator (see Sec. 5.2) using Java's parallel streaming library. We also used Java to store the polynomial factorization in memory.

Lastly, we stored the polynomial variables in a Postgres 9.5.5 database and stored the polynomial factorization in a text file. We perform all experiments on a 64bit Linux machine running Ubuntu 5.4.0. The machine has 120 CPUs and 1 TB of memory[8]. For the timing results, the Postgres database, which stores all the samples, also resides on this machine and has a shared buffer size of 250 GB.

7.2 Experimental Setup

For all our summaries, we ran our solver for 30 iterations or until the error was below 1×10^{-6} using the method presented in Sec. 3.3. Our summaries took under 1 day to compute with the majority of the time spent building the polynomial and solving for the parameters.

We evaluate EntropyDB on two real datasets as opposed to benchmark data to measure query accuracy in the presence of naturally occurring attribute correlations. The first dataset comprises information on flights in the United States from January 1990 to July 2015 [1]. We load the data into PostgreSQL, remove null values, and bin all real-valued attributes into equi-width buckets. We further reduce the size of the active domain to decrease memory usage and solver execution time by binning cities such that the two most popular cities in each state are separated and the remaining less popular cities are grouped into a city called 'Other'. We use equi-width buckets to facilitate transforming a user's query into our domain and to avoid hiding outliers, but it is future work to try different bucketization strategies. The resulting relation, `FlightsFine(fl_date, origin_city, dest_city, fl_time, distance)`, is 5 GB in size.

To vary the size of our active domain, we also create `FlightsCoarse(fl_date, origin_state, dest_state, fl_time, distance)`, where we use the origin state and destination state as flight locations. The left table in Fig. 8 shows the resulting active domain sizes.

The second dataset is 210 GB in size. It comprises N-body particle simulation data [27], which captures the state of astronomy simulation particles at different moments in time (snapshots). The relation `Particles(density, mass, x, y, z, grp, type, snapshot)` contains attributes that capture particle properties and a binary attribute, grp, indicating if a particle is in a cluster or not. We bucketize the continuous attributes (density, mass, and position coordinates) into equi-width bins. The right table in Fig. 8 shows the resulting domain sizes.

7.3 Query Accuracy

We first compare EntropyDB using our best statistic selection techniques of **COMPOSITE** and **2D** sort (see Sec. 7.5) to uniform and stratified sampling on the flights dataset. We use one percent samples, which require

	Flights Coarse	Flights Fine
fl_date (FD)	307	307
origin (OS/OC)	54	147
dest (DS/DC)	54	147
fl_time (ET)	62	62
distance (DT)	81	81
# possible tuples	4.5×10^9	3.3×10^{10}

	Particles
density	58
mass	52
x	21
y	21
z	21
grp	2
type	3
snapshot	3
# possible tuples	5.0×10^8

Figure 8: Active domain sizes. Each cell shows the number of distinct values after binning. Abbreviations shown in brackets are used in figures to refer to attribute names: e.g., OS stands for origin_state.

approximately 100 MB of space when stored in PostgreSQL. To approximately match the sample size, our largest summary requires only 600 KB of space in PostgreSQL to store the polynomial variables and approximately 200 MB of space in a text file to store the polynomial factorization. This, however, could be improved and compressed further beyond what we did in our prototype implementation.

We compute correlations on `FlightsCoarse` across all attribute pairs and identify the following pairs as having the largest correlations (C stands for "coarse"): 1C = (origin_state, distance), 2C = (destination_state, distance), 3CF = (fl_time, distance)[9], and 4C = (origin_state, destination_state). We use the corresponding attributes, which are also the most correlated, for the finer-grained relation and refer to those attribute pairs as 1F, 2F, and 4F.

Following the discussion in Sec. 6, we have two parameters to vary: B_a ("breadth") and B_s ("depth"). In order to keep the total number of statistics constant, we require that $B_a * B_s = 3000$. This threshold allows for the polynomial to be built and solved in under a day. Using this threshold, we build four summaries to show the difference in choosing statistics based solely on correlation (choosing statistics in order of most to least correlated) versus attribute cover (choosing statistics that cover the attributes with the highest combined correlation). The first summary, No2D, contains only 1D statistics. The next two, Ent1&2 and Ent3&4, use 1,500 statistics across the attribute pairs (1, 2) and (3, 4), respectively. The final one, Ent1&2&3, uses 1,000 statistics for the three attribute pairs (1, 2, 3). We do not include 2D statistics related to the flight date attribute because this attribute is relatively uniformly distributed and does not need a 2D statistic to correct for the MaxEnt's underlying uniformity assumption. Fig. 9 summarizes the summaries.

For sampling, we choose to compare with a uniform sample and four different stratified samples. We choose the stratified samples to be along the same attribute pairs as the 2D statistics in our summaries; i.e., pair 1 through pair 4.

To test query accuracy, we use the following query template:

```
SELECT A1, ... , Am, COUNT(*)
FROM R WHERE A1='v1' AND ... AND Am='vm'
GROUP BY A1, ... , Am
```

We test the approaches on 400 unique (A1, .., Am) values. We choose the attributes for the queries in a way that

---

[9]Pair 3 is the same for `FlightsCoarse` and `FlightsFine`

	MaxEnt Method	No2D	1&2	3&4	1&2&3
Pair 1	(origin, distance)		X		X
Pair 2	(dest, distance)		X		X
Pair 3	(time, distance)			X	X
Pair 4	(origin, dest)			X	

Figure 9: MaxEnt 2D statistics including in the summaries. The top row is the label of the MaxEnt method used in the graphs.
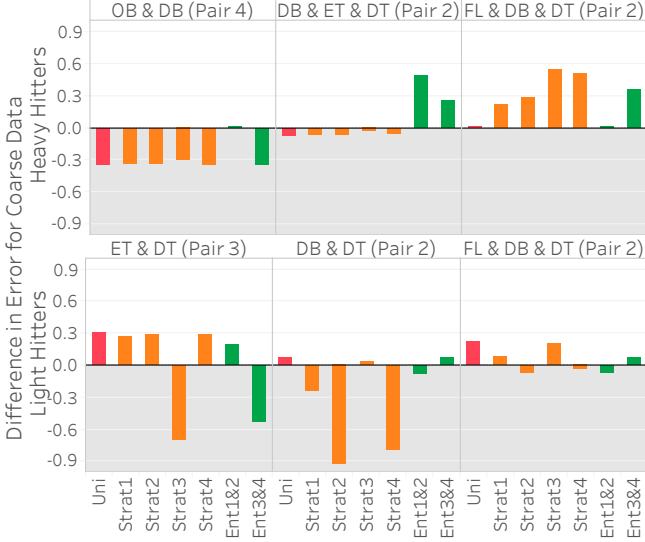
Figure 10: Query error difference between all methods and Ent1&2&3 over `FlightsCoarse`. The pair in parenthesis in the column header corresponds to the 2D statistic pair(s) used in the query template. For reference, pair 1 is (origin/OB, distance/DT), pair 2 is (dest/DB, distance/DT), pair 3 is (time/ET, distance/DT), and pair 4 is (origin/OB, dest/DB).

illustrates the strengths and weaknesses of EntropyDB. For the selected attributes, 100 of the values used in the experiments have the largest count (heavy hitters), 100 have the smallest count (light hitters), and 200 (to match the 200 existing values) have a zero true count (nonexistent/null values). To evaluate the accuracy of EntropyDB, we compute a $|true - est|/(true + est)$ (a measure of relative difference) on the heavy and light hitters. To evaluate how well EntropyDB distinguishes between rare and nonexistent values, we compute the F measure,

$$2 * \text{precision} * \text{recall}/(\text{precision} + \text{recall})$$

with

$$\text{precision} = \frac{|\{est_t > 0 \ : \ t \in \text{light hitters}\}|}{|\{est_t > 0 \ : \ t \in (\text{light hitters} \ \cup \ \text{null values})\}|}$$

and

$$\text{recall} = \frac{|\{est_t > 0 \ : \ t \in \text{light hitters}\}|}{100}.$$

We do not compare the execution time of EntropyDB to sampling for the flights data because the dataset is small, and the execution time of EntropyDB is, on average, below 0.5 seconds and at most 1 sec. Sec. 7.4.1 reports execution time for the larger data.

Fig. 10 (top) shows query error differences between all methods and Ent1&2&3 (i.e., average error for method X minus average error for Ent1&2&3) for three different heavy hitter queries over `FlightsCoarse`. Hence, bars above zero indicate that Ent1&2&3 performs better and vice versa. Each of the three query templates uses a different set of attributes that we manually select to illustrate different scenarios. The attributes of the query are shown in the column header in the figure, and any 2D statistic attribute-pair contained in the query attributes is in parentheses. Each bar shows the average of 100 query instances selecting different values for each template.

As the figure shows, Ent1&2&3 is comparable or better than sampling on two of the three queries and does worse than sampling on query 1. The reason it does worse on query 1 is that it does not have any 2D statistics over 4C, the attribute-pair used in the query, and 4C is fairly correlated. Our lack of a 2D statistic over 4C means we cannot correct for the MaxEnt's uniformity assumption. On the other hand, all samples are able to capture the correlation because the 100 heavy hitters for query 1 are responsible for approximately 25% of the data. This is further shown by Ent3&4, which has 4C as one of its 2D statistics, doing better than Ent1&2&3 on query 1.

Ent1&2&3 is comparable to sampling on query 2 because two of its 2D statistics cover the three attributes in the query. It is better than both Ent1&2 and Ent3&4 because each of those methods has only one 2D statistic over the attributes in the query. Finally, Ent1&2&3 is better than stratified sampling on query 3 because it not only contains a 2D statistic over 2C but also correctly captures the uniformity of flight date. This uniformity is also why Ent1&2 and a uniform sample do well on query 3. Another reason stratified sampling performs poorly on query 3 is because the result is highly skewed in the attributes of destination state and distance but remains uniform in flight date. The top 100 heavy hitter tuples all have the destination of 'CA' with a distance of 300. This means even a stratified sample over destination state and distance will likely not be able to capture the uniformity of flight date within the strata for 'CA' and 300 miles.

Fig. 10 (bottom) shows results for three different light hitter queries over `FlightsCoarse`. In this case, EntropyDB always does better than uniform sampling. Our performance compared to stratified sampling depends on the stratification and query. Stratified sampling outperforms Ent1&2&3 when the stratification is exactly along the attributes involved in the query. For example, for query 1, the sample stratified on pair 3 outperforms EntropyDB by a significant amount because pair 3CF is computed along the attributes in query 1. Interestingly, Ent3&4 and Ent1&2 do better than Ent1&2&3 on query 1 and query 2, respectively. Even though both of the query attributes for query 1 and query 2 are statistics in Ent1&2&3, Ent1&2 and Ent3&4 have more statistics and are thus able to capture more zero elements. Lastly, we see that for query 3, we are comparable to stratified sampling because we have a 2D statistic over pair 2C, and the other attribute, flight date, is relatively uniformly distributed in the query result.

We ran the same queries over the `FlightsFine` dataset and found *identical* trends in error difference. We therefore omit the graph.

An important advantage of our approach is that it more accurately distinguishes between rare values and nonexistent values compared with stratified sampling, which often does
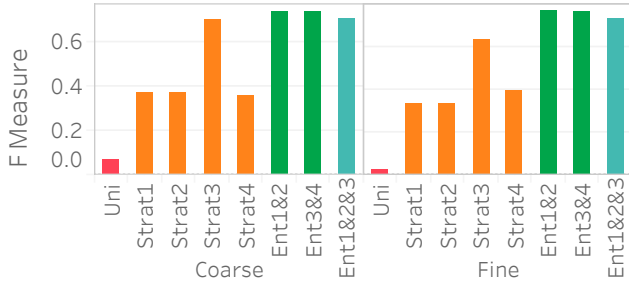
Figure 11: F measure for light hitters and null values over `FlightsCoarse` (left) and `FlightsFine` (right).

not have samples for rare values when the stratification does not match the query attributes. To assess how well our approach works on those rare values, Fig. 11 shows the average F measure over fifteen 2- and 3-dimensional queries selecting light hitters and null values.

We see that Ent1&2 and 3&4 have F measures close to 0.72, beating all stratified samples and also beating Ent1&2&3. The key reason why they beat Ent1&2&3 is that these summaries have the largest numbers of statistics, which ensures they have more fine grained information and can more easily identify regions without tuples. Ent1&2&3 has an F measure close to 0.69, which is slightly lower than the stratified sample over pair 3CF but better than all other samples. The reason the sample stratified over pair 3CF performs well is that the flight time attribute has a more skewed distribution and has more rare values than other dimensions. A stratified sample over that dimensions will be able to capture this. On the other hand, Ent1&2&3 will estimate a small count for any tuple containing a rare flight time value and will be rounded to 0.
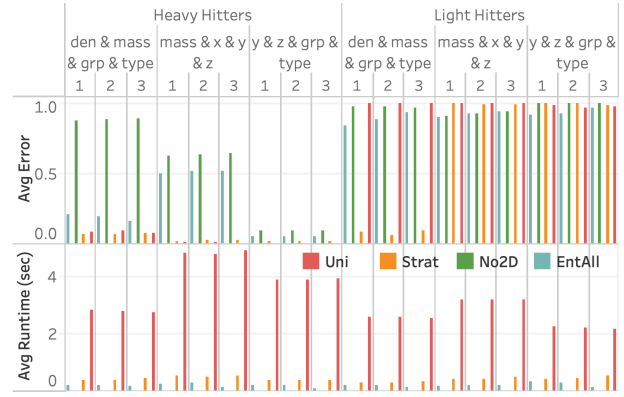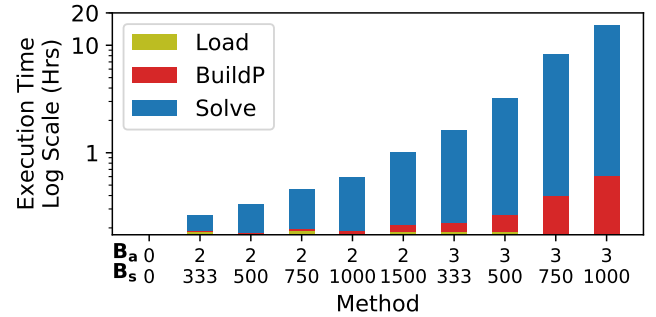
7.4 Execution Times

7.4.1 Scalability

To measure the performance of EntropyDB on large-scale datasets, we use three subsets of the 210 GB `Particles` table. We select data for one, two, or all three snapshots (each snapshot is approximately 70 GB in size). We build a 1 GB uniform sample for each subset of the table as well as a stratified sample over the pair density and group with the same sampling percentage as the uniform sample. We then build two MaxEnt summaries; EntNo2D uses no 2D statistics, and EntAll contains 5 2D statistics with 100 statistics over each of the most correlated attributes, not including snapshot. We do not use any presorting method for this experiment. We run a variety of 4D selection queries such as the ones from Sec. 7.3, split into heavy hitters and light hitters. We record the query accuracy and execution time.

Fig. 12 shows the query accuracy and execution time for three different selection queries as the number of snapshots increases. We see that EntropyDB consistently does better than sampling on query execution time, although both EntropyDB and stratified sampling execute queries in under one second. Stratified sampling outperforms uniform sampling because the stratified samples are generally smaller than their equally selective uniform sample.

In terms of query accuracy, sampling always does better than EntropyDB for the heavy hitter queries. This is expected because the bucketization of `Particles` is rela-

Figure 12: Query average error and execution time for three 4D selection queries on the `Particles` table. The stratified sample (orange) is stratified on (den, grp).

Figure 13: EntropyDB log scale execution times for loading the data, building the polynomial, and solving for the parameters for various configurations of B_a and B_s on `FlightsFine`.

tively coarse grained, and a 1 GB sample is sufficiently large to capture the heavy hitters. We do see that EntAll does significantly better than EntNo2D for query 1 because three of its five statistics are over the attributes of query 1 while only 1 statistic is over the attributes of queries 2 and 3. However, the query results of query 3 are more uniform, which is why EntNo2D and EntAll do well.

For the light hitter queries, none of the methods do well except for the stratified sample in query 1 because the query is over the attributes used in the stratification. EntAll does slightly better than stratified sampling on queries 2 and 3.

7.4.2 Solving Time

To show the data loading and model solving time of EntropyDB, we use `FlightsFine` and measure the time it takes for EntropyDB to read in a dataset from Postgres to collect statistics, to build the polynomial, and to solve for the model parameters for various B_a and B_s (see Fig. 13). When $B_a = 2$, we gather statistics over pair 1 and pair 2 (MaxEnt1&2), and when $B_a = 3$, we gather statistics over pair 1, 2, and 3 (MaxEnt3).)

We see that the overall polynomial building and solving execution time grows exponentially as B_a and B_s increase while the data loading time remains constant. The smallest model has a execution time of 10.5 minutes while largest model (MaxEnt3) has a execution time of 15.4 hours. The
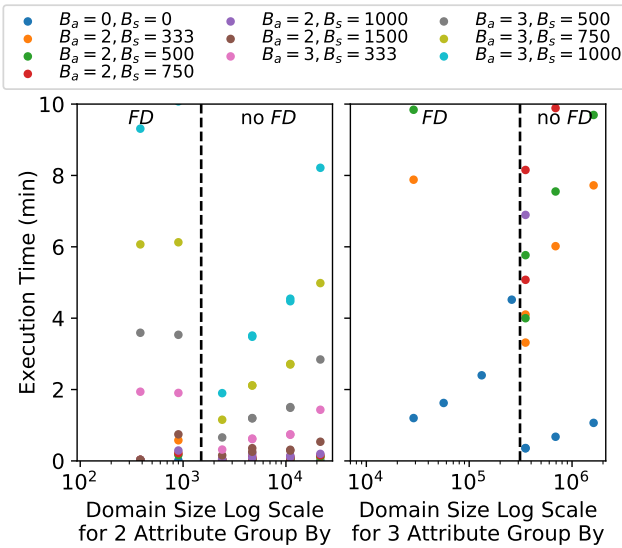
Figure 14: Log scale query execution times for 2- and 3-dimensional group-by queries versus size of query's active domain on `FlightsFine` for various configurations of B_a and B_s. The dashed line demarcates queries with `fl_date` as a group-by attribute and those that do not.

experiment further demonstrates that B_a impacts execution time more than B_s. The method with $B_a = 2, B_s = 750$ has a faster execution time than the method with $B_a = 3, B_s = 500$ even though the total number of statistics, 1,500 in both, is the same.

Note that the data loading time (yellow) will increase as the dataset gets larger, but once all the histograms and statistics are computed, the time to build the polynomial and solver time are independent of the original data size; they only depend on the model complexity.

7.4.3 Group By Queries

To further expand on execution time results, we measure the execution time to compute eight various 2- and 3-dimensional group-by queries instead of single point queries (sixteen group-by queries in total) to show how the execution time depends on the active domain. As EntropyDB can only issue a single point query at a time (the query evaluation is already parallelized), the group-by queries are run as sequences of point queries over the domain of the query attributes.

Fig. 14 shows a scatter plot of the query domain size versus the execution time for 2- and 3-dimensional group-by queries for the same models as used in Sec. 7.4.2 (i.e. $B_a = 0, 2, 3$ and B_s varying form 333 to 1500). If a model takes longer than 10 minutes to compute a group-by query, we terminate its execution. Each color represents a different combination of B_a and B_s. Note that running a 3-dimensional group-by query on Postgres on the full `FlightsFine` can take up to 17 minutes.

The overall trend we see is that models with a larger B_a are slower to execute, and for models with the same B_a, larger B_s is slower. For example, the average execution for 2-dimensional group-by queries for $B_a = 2$ is 8 seconds while it is 87 seconds for $B_a = 3$. This is not surprising and matches the results from Fig. 13. We again see that $B_a =$

$2, B_s = 750$ is faster than $B_a = 3, B_s = 500$ even though the total number of statistics is the same.

Some of the large models had an execution time of longer than 10 minutes for some of the 3-dimensional queries, which is why their scatter point is not shown on all queries. Even though their execution time was more than 10 minutes, each individual point query still ran in under a second.

The results also show a surprising trend in that the execution time dips after the black dashed line and then starts slowing increasing again. This dashed black line demarcates queries containing the `fl_date` attribute, the one attribute not included in any 2-dimensional statistic. Note that because the active domain of FD is small, the smaller domain queries happen to contain FD, but the size of the domain is independent of the dip in the execution time.

This unintuitive result is explained by the optimizations in Sec. 5.2. By using bit vectors and maps to indicate which attributes and variables are contained in a polynomial subterm, we can quickly decide if that subterm needs to be set to zero or not for query evaluation. This mainly improves evaluation of correction subterms (i.e. $(\delta - 1)$ times 1D sums) because the 1D sums contain subsets of the active domain and are more likely overlap with the variables that can be set to zero. The more quickly we can decide if a subterm is zero, the faster the evaluation.

For example, take the polynomial in Fig. 2. If we are evaluating a query for $A = 155 \wedge B = 700 \wedge C = 700$, then all other α, β, and γ variables need to be set to zero except α_{155}, β_{700}, and γ_{700}. This means the polynomial sums on lines 3, 5, 6, and 7 can all be set to zero without having to evaluate each individual subterm on those lines because α_{155}, β_{700}, and γ_{700} are not contained in any of those subterms and A, B, and C attributes are meant to be zero.

The FD attribute being one of the group-by attributes indicates that all variables representing FD except for the one being selected can be set to zero. However, as FD is not part of a statistic, there are no correction terms being multiplied by subsets of the FD active domain. Therefore, there are fewer chances to set a subterm to zero, meaning the overall query execution is slower.

This evaluation presents and interesting tradeoff between model size, statistic attributes, and query execution. One the one hand, a larger model will take longer to run, in general. On the other hand, more statistics allow for more zero setting optimizations in query evaluation. We also see that while EntropyDB can handle 2-dimensional group-by queries, especially if $B_a = 2$, it struggles to perform for 3-dimensional ones. However, as the strength of EntropyDB is in querying for light hitters, EntropyDB will miss fewer groups than sampling techniques which are more impacted by heavy hitters. We leave it as future work further optimize large domain group-by queries.

7.5 Statistic Selection

7.5.1 Selection Technique

We evaluate the three different statistic selection heuristics, described in Sec. 6.1, on `FlightsCoarse` restricted to the attributes (date, time, distance). We gather statistics using the three different techniques and using different budgets on the attribute pair (time, distance). There are 5,022 possible 2D statistics, 1,334 of which exist in
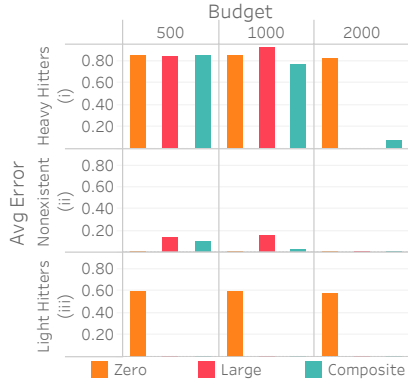
Figure 15: Illustration of query accuracy versus budget for the three different heuristics and three different selections: (i) selecting 100 heavy hitter values, (ii) selecting 200 nonexistent values, and (iii) selecting 100 light hitter values.

`FlightsCoarse`. We evaluate the accuracy of the resultant count of the query

```
SELECT time, dist, COUNT(*)
FROM Flights WHERE time = x AND dist = y
GROUP BY time, dist
```

for 100 heavy hitter (x, y) values, 100 light hitter (x, y) values, and 200 random (x, y) nonexistent/zero values. We choose 200 zero values to match the 100+100 heavy and light hitters.

Fig. 15 (i) plots the query accuracy versus method and budget for 100 heavy hitter values. Both **LARGE** and **COMPOSITE** achieve almost zero error for the larger budgets while **ZERO** gets around 60 percent error no matter the budget.

(ii) plots the same for nonexistent values, and clearly **ZERO** does best because it captures the zero values first. **COMPOSITE**, however, gets a low error with a budget of 1,000 and outperforms **LARGE**. Interestingly, **LARGE** does slightly worse with a budget of 1,000 than 500. This is a result of the final value of P being larger with a larger budget, and this makes our estimates slightly higher than 0.5, which we round up to 1. With a budget of 500, our estimates are slightly lower than 0.5, which we round down to 0.

Lastly, (iii) plots the same for 100 light hitter values, and while **LARGE** eventually outperforms **COMPOSITE**, **COMPOSITE** gets similar error for all budgets. In fact, **COMPOSITE** outperforms **LARGE** for a budget of 1,000 because **LARGE** predicts that more of the light hitter values are nonexistent than it does with a smaller budget as less weight is distributed to the light hitter values.

Unsurprisingly, we see that **COMPOSITE** is the best method to use across all queries. However, the **COMPOSITE** method is more complex and takes more time to compute. We do learn that if heavy hitter queries are the only relevant queries in a particular workload, it is unnecessary to use the **COMPOSITE** method as **LARGE** does just as well. Also, **ZERO** is the best if existence queries are the most important (e.g. if determining set containment). So while **COMPOSITE** is best for a handling a variety of queries, it may not be necessary, depending on the query workload.

(a) Heavy Hitters No Sort

(b) Heavy Hitters **2D** Sort

(c) Light Hitters No Sort

(d) Light Hitters **2D** Sort

(e) F Measure No Sort

(f) F Measure **2D** Sort

Figure 16: (a, b) Error over 2D heavy hitter queries, (c, d) error over 2D light hitter queries, and (e, f) F measure over 2D light hitter and null value queries across different MaxEnt methods over `FlightsCoarse` and `FlightsFine` using no sort (left side) and 2D sort (right side).

7.5.2 Statistic Accuracy

We now investigate, in more detail, how the different 2D statistic attribute choices and how presorting the matrix impacts query accuracy. We look at the query accuracy of the four different MaxEnt methods used in Fig. 10 using both **2D** sort and no sort. We also include the MaxEnt method No2D for comparison although it does not use any sorting. The no sort technique maintains the natural ordering of the domains. We use `FlightsCoarse` and `FlightsFine` and the query templates from Sec. 7.3. We run six different two-attribute selection queries over all possible pairs of the attributes covered by pair 1 through 4; i.e., origin, destination, time, and distance. We select 100 heavy hitters, 100 light hitters, and 200 null values.

Fig. 16 shows the average error for the heavy hitters and the light hitters and shows the average F measure across the six queries. The left side shows the error when no presorting is used, and the right side shows the error when **2D** sort is used. We first consider the different attribute selections. We see that the summary with more attribute pairs but fewer buckets (more "breadth"), Ent1&2&3, does best on the heavy hitters. On the other hand, for the light hitters, we see that the summary with fewer attribute pairs but more buckets (more "depth") and still covers the attributes, Ent3&4, does best. Ent3&4 doing better than Ent1&2 implies that choosing the attribute pairs that cover the at-

tributes yields better accuracy than choosing the most correlated pairs because even though Ent1&2 has the most correlated attribute pairs, it does not have a statistic containing flight time. Lastly, Ent1&2&3 does best on the heavy hitter queries yet slightly worse on the light hitter queries because it does not have as many buckets as Ent1&2 and Ent3&4 and can thus not capture as many regions in the active domains with no tuples.

When considering presorting the data, we see that **2D** sort does not have any significant impact on heavy hitter accuracy, with an improvement on the order of 0.001. For light hitters, we see a slight average error improvement using **2D** sort, and for the F measure, we see a slight decrease in measure. Ent2&3 has the largest improvement in light hitter query error because of the improvement in resorting pair 3 (distance and time) along with its large K-D tree leaf budget. Ent2&3 has 1,500 K-D tree leaves which is enough to capture the 1,334 nonzero values of pair 3.

The decrease in F measure and limited improvement for query accuracy is best explained by looking at the sorted and unsorted frequency heatmaps and K-D trees of the pair 2C for Ent1&2&3 on `FlightsCoarse`, shown in Fig. 17. We see that the average K-D tree error does, in fact, decrease, which should indicate an improvement in accuracy and F measure. However, upon closer inspection of the K-D tree leaves, we see that the sorted tree actually has put more zeros in leaves with some small, nonzero values. This, in turn, causes MaxEnt1&2&3 to believe those zeros actually exist, therefore decreasing the F measure. This result implies that improving K-D tree error is not always enough to guarantee a high F measure because every zero that is in a leaf with some nonzero value will be misclassified as existing.

8. DISCUSSION

8.1 Future Work

The above evaluation shows that EntropyDB is competitive with stratified sampling overall and better at distinguishing between infrequent and absent values. Importantly, unlike stratified sampling, EntropyDB's summaries permit multiple 2D statistics. Further, as EntropyDB is based on modeling the data, it does not actually need access to the original, underlying data. If a data scientist only has access to, for example, various 2-dimensional histogram queries of the entire dataset, EntropyDB would still be able to build a model of the data and answer queries. Sampling would only be able to handle queries that are directly over one of the histograms. The main limitations of EntropyDB are the dependence on the size of the active domain, correlation-based 2D statistic selection, manual bucketization, and limited query support.

To address the first problem, our future work is to investigate using standard algebraic factorization techniques on non-materializable polynomials. By further reducing the polynomial size, we will be able to handle larger domain sizes. We also will explore using statistical model techniques to more effectively decompose the attributes into 2D pairs, similar to [19]. To no longer require bucketizing categorical variables (like city), we will research hierarchical polynomials. These polynomials will start with coarse buckets (like states), and build separate polynomials for buckets that require more detail. This may require the user to wait while a
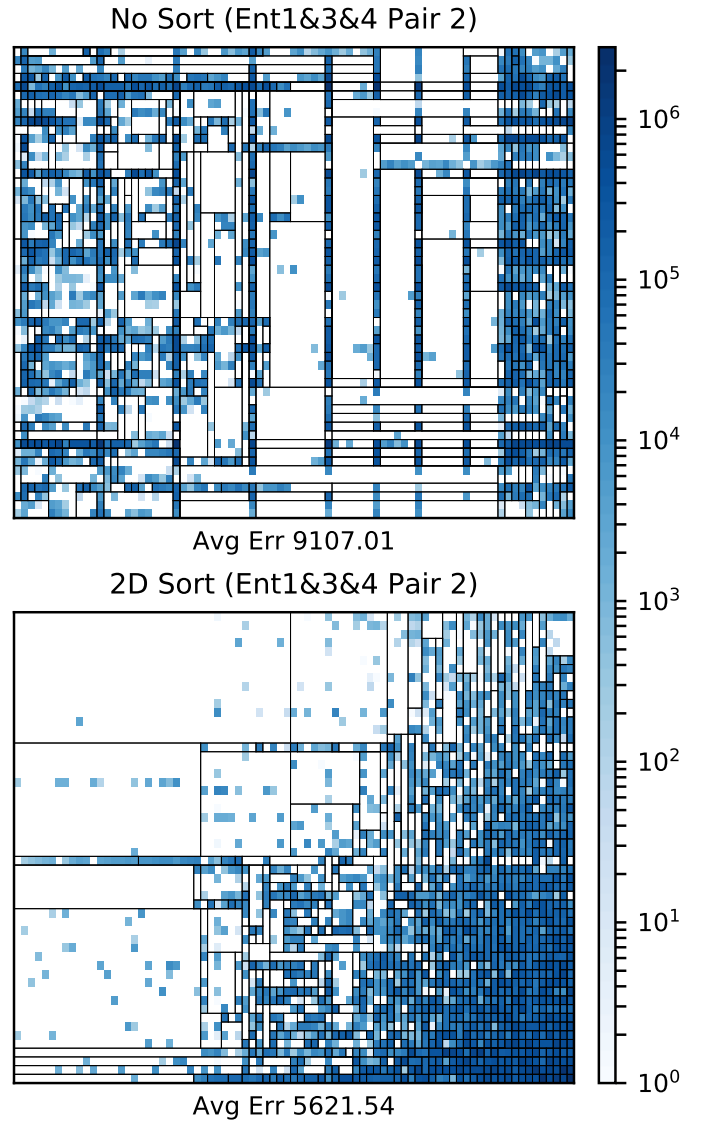

Figure 17: Plots showing the frequency heatmap of the pair 2 attributes of MaxEnt1&3&4 of **FlightsCoarse** that is unsorted (left) and sorted using the 2D sort algorithm (right). The average K-D tree error is shown below.

new polynomial is being loaded but would allow for different levels of query accuracy without sacrificing polynomial size.

Addressing our queries not reporting error is non-trivial and requires combining the errors in the statistics with the errors in the model parameters with the errors in making the uniformity assumption for the attributes not covered by a statistic. Our future work will be to understand how the error depends on the each of these facets and developing an error equation that propagates these errors through polynomial evaluation.

8.2 Handling Joins and Data Updates

When building our MaxEnt summary, we assume there was only a single relation being summarized and the underlying data is not updated. We now discuss two extensions of our summarization technique to address both of these

assumptions.

8.2.1 Joins

In Sec. 3, we introduce the MaxEnt model over a single, universal (pre-joined) relation R and an instance $\mathbf{I}$ of R. Now, suppose the data we want to summarize consists of r relations, $R_1, \ldots, R_r$, each with an associated instance $\mathbf{I}_1, \ldots, \mathbf{I}_r$. To describe our approach, we assume each R_i joins with R_{i+1} by an equi-join on attribute $A_{j_{i,i+1}}$; i.e. $R = R_1 \bowtie_{R_1.A_{j_{1,2}} = R_2.A_{j_{1,2}}} R_2 \bowtie \ldots \bowtie R_r$. Our technique can easily be extended to work for multiple equi-join attributes, but for simplicity, we describe the approach for a single join attribute. Let $R(A_1, \ldots, A_m)$ be the global schema of $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_r$ with active domains as described in Sec. 3.

The simplest approach to handle joins is to join all relations and build a summary over the universal relation. Once the summary is built, the universal relation can be removed. While this summary can now handle queries over R, it requires R to be computed once, which can be an expensive procedure.

Our approach is to build a separate MaxEnt data summary for each instance: $\{(P_1, \{\alpha_j\}_1, \Phi_1), \ldots, (P_r, \{\alpha_j\}_r, \Phi_r)\}$. A linear query $\mathbf{q}$ with associated predicate $\pi_{\mathbf{q}}$ over R is answered by iteration over the distinct values in the join attributes; i.e.

$$\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle] = \sum_{d_1 \in D_{j_{1,2}}} \ldots \sum_{d_{r-1} \in D_{j_{r-1,r}}} \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_1 \rangle] \ldots \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_r \rangle]$$

$$\text{s.t. } \pi_{\mathbf{q}'} = \pi_{\mathbf{q}} \wedge (R_1.A_{j_{1,2}} = d_1) \wedge (R_2.A_{j_{1,2}} = d_1) \wedge$$
$$\ldots \wedge (R_r.A_{j_{r-1,r}} = d_{r-1}).$$

where $\mathbf{q}'$ is the linear query associated with $\pi_{\mathbf{q}'}$ and $R_i.A_j$ denotes attribute A_j in relation R_i. We abuse notation slightly in that $\mathbb{E}[\langle \mathbf{q}', \mathbf{I}_i \rangle]$ is the answer to $\mathbf{q}'$ projected on to the attributes of R_i (i.e. setting $\rho \equiv true$ for attributes not in R_i). Note that if $\mathbf{q}$ is only over a subset of relations, then the summation only needs to be over the distinct join values of the relations in the query.

While this method will return an approximate answer, it does rely on iteration over the active domain of the join attributes, which, as we shown in Fig. 14, can be expensive for larger domain sizes. However, for each relation R_i, if we modify the statistic constraints associated with the 1D statistics of the join attribute $A_{j_{i,i+1}}$, we can improve runtime by decreasing the number of iterations in the summation.

At a high level, before learning multi-dimensional statistics, for each $\ell \in J_{j_{i,i+1}}$ (i.e. each 1D statistic index for attribute $A_{j_{i,i+1}}$), we replace $(\mathbf{c}_\ell, s_\ell)$ by $(\mathbf{c}_\ell, \bar{s})$ where $\bar{s}$ is the average s_ℓ value of a group of statistics in $J_{j_{i,i+1}}$. This is similar to building a **COMPOSITE** statistic over $A_{j_{i,i+1}}$ except instead of replacing each individual statistic by the composite, we are modifying the constraint for each statistic. We do do not replace the 1D statistics because we still want to be able to query at the level of an individual tuple. As our querying technique is equivalent to derivation, if we remove the fine-grained 1D statistics, there is nothing to derivate by if a query is issued over a 1D statistic.

Specifically, with $B'_s \leq B_s$ as the budget for the 1D statistic, suppose we learn that $\{g_k^{i,i+1} = [l_k^i, u_k^i] : k = 1, B'_s\}$ is the optimal set of boundaries for join attribute $A_{j_{i,i+1}}$ from relation R_i to R_{i+1}. These can be learned with the K-D tree

method in Sec. 6 by sorting and then repeatedly splitting on the single axis until the budget B'_s is reached. We then apply the same bounds of $\{[l_k^i, u_k^i] : k = 1, B'_s\}$ on any multi-dimensional statistic covering $A_{j_{i,i+1}}$ in R_i and R_{i+1} and the 1D statistic covering $A_{j_{i,i+1}}$ in R_{i+1} (see Example 8.1). As this boundary is learned before multi-dimensional statistics are built, when we build a 2-dimensional statistic covering $A_{j_{i,i+1}}$, we seed the K-D tree with the $A_{j_{i,i+1}}$ axis splits of $\{g_k^{i,i+1} : k = 1, B'_s\}$ and repeatedly split on the other axis until reaching our budget B_s.

Using this transfer boundary technique and rewriting the summation, we can answer queries over joins by iterating over a single point in each range boundary rather than all individual values. The following example gives intuition as to how this boundary transfer works.

EXAMPLE 8.1. *Suppose we have two relations $R(A, B)$ and $S(B, C)$ with instances $\mathbf{I}_R$ and $\mathbf{I}_S$ where each attribute has an active domain size of 3. We also have a query $\mathbf{q}$ with predicate $\pi_{\mathbf{q}} = (A = a_1 \wedge C = c_1)$ over $R \bowtie S$. Let each relation build a data summary $(P, \{\alpha_j\}, \Phi)$ using a 1D composite statistic over B and a 2D statistic over their two attributes with $B_s = 4$ and $B'_s = 2$. Lastly, for the composite statistic, suppose we learn that the optimal boundaries for B are $[b_1, b_2]$ and $[b_3]$, meaning the statistics over B will be*

$$(B = b_1, \ (s_{b_1} + s_{b_2})/2)$$
$$(B = b_2, \ (s_{b_1} + s_{b_2})/2)$$
$$(B = b_3, \ s_{b_3})$$

where s_{b_i} is the number of tuples where $B = b_i$. Note that the constraint for b_1 and b_2 is the same which implies $n\beta_1 \partial P / P \partial \beta_1 = n\beta_2 \partial P / P \partial \beta_2$ by Eq. 1.

*By our naïve strategy, the answer to $\mathbf{q}$ is*

$$\mathbb{E}[\langle \mathbf{q}, \mathbf{I_R} \bowtie \mathbf{I_S} \rangle] = \sum_{b \in [b_1, b_3]} \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_R \rangle] \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_S \rangle]$$

$$\text{s.t. } \pi_{\mathbf{q}'} = (A = a_1 \wedge C = c_1 \wedge B = b).$$

This can be rewritten in terms of polynomial derivation as

$$= \frac{n_R \alpha_1 \beta_1}{P_R} \frac{\partial P_R}{\partial \alpha_1 \partial \beta_1} \frac{n_S \beta_1 \gamma_1}{P_S} \frac{\partial P_S}{\partial \beta_1 \partial \gamma_1}$$
$$+ \frac{n_R \alpha_1 \beta_2}{P_R} \frac{\partial P_R}{\partial \alpha_1 \partial \beta_2} \frac{n_S \beta_2 \gamma_1}{P_S} \frac{\partial P_S}{\partial \beta_2 \partial \gamma_1}$$
$$+ \frac{n_R \alpha_1 \beta_3}{P_R} \frac{\partial P_R}{\partial \alpha_1 \partial \beta_3} \frac{n_S \beta_3 \gamma_1}{P_S} \frac{\partial P_S}{\partial \beta_3 \partial \gamma_1}.$$

Consider two cases: when the other 2-dimensional statistics have the same boundaries on B and when they do not. For the first case, let the 2D statistics over R and S be as shown by the rectangles all in black with the red line for S.
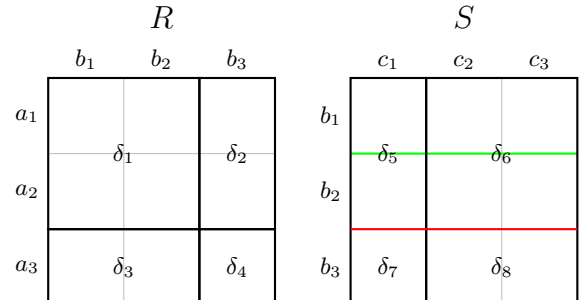
Using these statistics, $\mathbb{E}[\langle \mathbf{q}, \mathbf{I_R} \bowtie \mathbf{I_S} \rangle]$ now becomes

$$= \frac{n_R \alpha_1 \beta_1}{P_R} \delta_1 \frac{n_S \beta_1 \gamma_1}{P_S} \delta_5 + \frac{n_R \alpha_1 \beta_2}{P_R} \delta_1 \frac{n_S \beta_2 \gamma_1}{P_S} \delta_5$$
$$+ \frac{n_R \alpha_1 \beta_3}{P_R} \delta_2 \frac{n_S \beta_3 \gamma_1}{P_S} \delta_7$$
$$= 2 \frac{n_R \alpha_1 \beta_1}{P_R} \delta_1 \frac{n_S \beta_1 \gamma_1}{P_S} \delta_5 + \frac{n_R \alpha_1 \beta_3}{P_R} \delta_2 \frac{n_S \beta_3 \gamma_1}{P_S} \delta_7$$

where the second line follows because $n\beta_1 \partial P / P \partial \beta_1 = n\beta_2 \partial P / P \partial \beta_2$. Notice how instead of summing over all distinct values in B, we are summing over B'_s values.

In we had statistics over S that were the rectangles in black with the green line (the boundaries on B do not match those of the composite 1D statistic), $\mathbb{E}[\langle \mathbf{q}, \mathbf{I_R} \bowtie \mathbf{I_S} \rangle]$ would be

$$= \frac{n_R \alpha_1 \beta_1}{P_R} \delta_1 \frac{n_S \beta_1 \gamma_1}{P_S} \delta_5 + \frac{n_R \alpha_1 \beta_2}{P_R} \delta_1 \frac{n_S \beta_2 \gamma_1}{P_S} \delta_6$$
$$+ \frac{n_R \alpha_1 \beta_3}{P_R} \delta_2 \frac{n_S \beta_3 \gamma_1}{P_S} \delta_7$$

which does not simplify.

Formally, suppose we only use the transfer boundary technique for $A_{j_{r-1,r}}$, the last join attribute; i.e. we make the 1D composite statistic boundaries of $A_{j_{r-1,r}}$ the same in R_{r-1} and R_r. Using $|g_k^{i,i+1}| = u_k^i - l_k^i + 1$ (the size of the range), we can rewrite $\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle]$ as

$$\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle] = \sum_{d_1 \in D_{j_{1,2}}} \cdots \sum_{d_{r-1} \in D_{j_{r-1,r}}} \prod_{i=1}^{r} \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_i \rangle]$$

$$= \sum_{d_1 \in D_{j_{1,2}}} \cdots \sum_{g_k^{r-1,r}} \sum_{d_{r-1} \in g_k^{r-1,r}} \prod_{i=1}^{r} \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_i \rangle]$$

$$= \sum_{d_1 \in D_{j_{1,2}}} \cdots \sum_{g_k^{r-1,r}} \mathbb{E}[\langle \mathbf{q}'_r, \mathbf{I}_r \rangle] \sum_{d_{r-1} \in g_k^{r-1,r}} \prod_{i=1}^{r-1} \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_i \rangle]$$

$$= \sum_{d_1 \in D_{j_{1,2}}} \cdots \sum_{g_k^{r-1,r}} \Big[|g_k^{r-1,r}| * \mathbb{E}[\langle \mathbf{q}'_{r-1}, \mathbf{I}_{r-1} \rangle]$$

$$* \mathbb{E}[\langle \mathbf{q}'_r, \mathbf{I}_r \rangle] * \prod_{i=1}^{r-2} \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_i \rangle] \Big]$$

s.t. $\pi_{\mathbf{q}'_{r-1}} = \pi \wedge (R_1.A_{j_{1,2}} = d_1) \wedge \ldots \wedge (R_r.A_{j_{r-1,r}} = true)$
$$\wedge (R_{r-1}.A_{j_{r-1,r}} = D_{j_{r-1,r}}[l_k^i])$$
$\pi_{\mathbf{q}'_r} = \pi \wedge (R_1.A_{j_{1,2}} = d_1) \wedge \ldots \wedge (R_{r-1}.A_{j_{r-1,r}} = true)$
$$\wedge (R_r.A_{j_{r-1,r}} = D_{j_{r-1,r}}[l_k^i]).$$

The step from line one to line two is replacing the sum over $d_{r-1} \in D_{j_{r-1,r}}$ to a sum over boundaries $\{g_k^{r-1,r}\}$ and a sum over distinct values in the boundary. In line three, we pull out the query over $\mathbf{I}_r$ because the answer for $\mathbb{E}[\langle \mathbf{q}', \mathbf{I}_r \rangle]$ is the same for each $d_{r-1} \in g_k^{r-1,r}$ as they use the same composite statistic. Therefore, we pull out the query and modify the query's predicate to be over the lower boundary value (any value in the boundary would produce equivalent results).

In line four, we perform the same trick and pull out the query over $\mathbf{I}_{r-1}$ because $\mathbb{E}[\langle \mathbf{q}', \mathbf{I}_{r-1} \rangle]$ will also be the same for each $d_{r-1} \in g_k^{r-1,r}$. Lastly, because $\prod_{i=1}^{r-2} \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_i \rangle]$ is independent of R_{r-1} as it does not contain $\mathbf{I}_{r-1}$, we can also pull it out of the sum. At the end, we get a summation

---

Algorithm 4 Update Model

```
for (Δt) do
    Φ = updateStats(Φ, Δt)
    if (not {α_j} being updated) do
        if timeToRebuild do
            (P,{α_j},Φ) = rebuildModel(R)
        else
            {α_j} = updateParams(Φ, {α_j})
```

---

over the value one that repeats $|g_k^{r-1,r}|$ times. This sum rewriting trick can be applied to all attributes with shared boundaries on all statistics covering the join attributes.

By performing this boundary transfer trick, we have replaced the sum for distinct values of $A_{j_{r-1,r}}$ with the sum over lower boundary points of $\{g_k^{i,i+1}\}$. We can repeat this boundary transfer for any of the dense distinct join values to make the final join algorithm efficient. Note that this technique does lose accuracy as we are no longer building building fine-grained 1D statistics over the join attributes and are using potentially suboptimal boundaries for other multi-dimensional statistics.

8.2.2 Updates

Another key assumption made in Sec. 3 is that the data being summarized is read only and not updated. If we relax that assumption and let the underlying data change, our model needs to be updated, too. We make the assumption that data updates are represented as single tuple additions or deletions. For example, a value change can be represented as a tuple deletion followed by a tuple addition. Alg. 4 describes our update technique.

The intuition behind our algorithm is that as updates come in, it is satisfactory to initially only update the polynomial parameters $\{\alpha_j\}$ while keeping the statistic predicates the same. However, as the data continues to be updated, the underlying correlations and relationships of the attributes may change, meaning the statistic predicates are no longer optimal. When this occurs, the entire summary needs to be rebuilt. Ideally, the rebuilding would happen overnight or when the summary is not in high demand.

Out algorithm works as follows. For each tuple update, updateStats modifies s_j for each predicate π_j that t satisfies. s_j increases or decreases by one depending of it t is being added or removed. It is important to realize that updateStats does *not* update the predicates defining the statistics, just the predicate values.

After the statistics are updated, we check if either updateParams or rebuildModel is currently running or in progress. If it is, we move on to the next update, effectively batching our changes. If no update is in progress, we update or rebuild our model. updateParams simply updates the polynomial parameters $\{\alpha_j\}$ by running Alg. 1 initialized by the last solved for parameters. By initializing our model at the last know solution, we decrease convergence time because many of the parameters are already solved for and do not need to change. In contrast to simply updating our parameters, rebuildModel starts from scratch and regenerates the statistics, polynomial, and parameters.

The final method to discuss, timeToRebuild, decides whether to update or rebuild the model. There are numerous different ways to defining timeToRebuild, and we give three such possibilities below.

- When the number of tuple updates reaches some predefined threshold B.
- When the system does not have many users, meaning there is more compute power to rebuild the summary.
- When attribute correlations are not accurately represented in Φ. i.e. when some attribute pair in Φ is uniformly distributed or when some attribute pair in R is correlated but not included in Φ.

8.3 Connection to Probabilistic Databases

At a high level, EntropyDB learned a probability distribution of the data so that each possible instance has some associated probability of existing. Since this possible world semantics is the same semantics as used by probabilistic databases, how does EntropyDB relate to probabilistic databases [43, 18]?

Recall that probabilistic databases store uncertain data, and, like EntropyDB, represent the probability of a tuple as $\Pr(t) = \sum_{I \in PWD | t \in I} \Pr(I)$. The uncertainty in the data arises from the application such as data extraction, data integration, or data cleaning. Probabilistic databases are commonly stored as tuple independent (TI) databases where each tuple has an associated marginal probability and is an independent probabilistic event.

EntropyDB, on the other hand, does not store the marginal probabilities. Recall that for EntropyDB, if we let $\mathbf{q} = \mathbf{t}_\ell$ for some tuple t_ℓ, then, from Eq. 11, we can calculate the expected number of times tuple t_ℓ appears in an instance. If we divide by n, we calculate the expected percent of times tuple t_ℓ appears in an instance. This, in general, is not the same as the marginal probability. It turns out that when $n = 1$, the two are equivalent.

LEMMA 8.2. *For some tuple t_ℓ and associated linear query $\mathbf{t}_\ell$ (query with 1 in the place of tuple t_ℓ and 0 elsewhere), when $n = 1$, $\mathbb{E}[\langle \mathbf{t}_\ell, \mathbf{I} \rangle]$ is the marginal probability of a tuple; i.e., $\Pr(t_\ell \in I)$.*

PROOF. We will use the same trick as in Sec. 3.2 by extending the polynomial with a new variable $\beta = 1$ representing the query $\mathbf{t}_\ell$. Denote this extended polynomial as $P_{\mathbf{t}_\ell}$. Following Eq. 10, we get

$$
\begin{aligned}
(P_{\mathbf{t}_\ell})^n &= \left(\sum_{i=1,d} \prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{t}_i \rangle} \beta^{\langle \mathbf{t}_\ell, \mathbf{t}_i \rangle} \right)^n \\
&= \left(\prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{t}_\ell \rangle} \beta + \sum_{\substack{i=1,d \\ i \neq \ell}} \prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{t}_i \rangle} \beta^{\langle \mathbf{t}_\ell, \mathbf{t}_i \rangle} \right)^n \\
&= \left(\left(\beta \frac{\partial P_{\mathbf{t}_\ell}}{\partial \beta} \right) + \left(P_{\mathbf{t}_\ell} - \beta \frac{\partial P_{\mathbf{t}_\ell}}{\partial \beta} \right) \right)^n .
\end{aligned}
\tag{25}
$$

Note that $\beta^{\langle \mathbf{t}_\ell, \mathbf{t}_i \rangle} = 0$ when $\ell \neq i$. We use this rewriting in finding the marginal probability.

$$
\begin{aligned}
\Pr(t_\ell \in I) &= 1 - \Pr(t_\ell \notin I) \\
&= 1 - \frac{1}{(P_{\mathbf{t}_\ell})^n} \sum_{I:t_\ell \notin I} \prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{I} \rangle} \beta^{\langle \mathbf{t}_\ell, \mathbf{I} \rangle} \\
&= 1 - \frac{1}{(P_{\mathbf{t}_\ell})^n} \left(\sum_{\substack{i=1,d \\ i \neq \ell}} \prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{t}_i \rangle} \beta^{\langle \mathbf{t}_\ell, \mathbf{t}_i \rangle} \right)^n \\
&= 1 - \frac{1}{(P_{\mathbf{t}_\ell})^n} \left(P_{\mathbf{t}_\ell} - \beta \frac{\partial P_{\mathbf{t}_\ell}}{\partial \beta} \right)^n \\
&= 1 - \left(1 - \frac{\beta}{P_{\mathbf{t}_\ell}} \frac{\partial P_{\mathbf{t}_\ell}}{\partial \beta} \right)^n
\end{aligned}
$$

The third line follows a similar proof as Eq. 6. From Eq. 11 and since $\beta = 1$ and $n = 1$, we get

$$
\begin{aligned}
\Pr(t_\ell \in I) &= \frac{1}{P_{\mathbf{t}_\ell}} \frac{\partial P_{\mathbf{t}_\ell}}{\partial \beta} \\
&= \mathbb{E}[\langle \mathbf{t}_\ell, \mathbf{I} \rangle]
\end{aligned}
$$

$\square$

It is important to note that although we can calculate the marginal probability, the probabilities are not independent; i.e., we do not have a TI probabilistic database.

8.4 Connection to Graphical Models

The Principle of Maximum Entropy is well studied, and it is known that the maximum entropy solution with marginal constraints (i.e., COUNT(*) constraints) is equivalent to the maximum likelihood solution for exponential family models [37, 44, 47, 28]. This can be seen by transforming Eq. 2 into exponential form

$$
\Pr(I) = \exp \left(\left(\sum_{j=1}^{k} \theta_j \phi_j(I) \right) \right.
\tag{26}
$$

$$
\left. - \log \sum_{I \in PWD} \left(\exp \left(\sum_{j=1}^{k} \theta_j \phi_j(I) \right) \right) \right).
\tag{27}
$$

Further, this exponential form is equivalent to the probability distribution defined over an exponential family Markov Network or, more generally, factor graph [49]. Markov networks are factor graphs where the factors representing the parameterization of the probability distribution are defined solely on cliques in the graph.

This connection implies that we can use MLE techniques to solve for the parameters θ_j, and, in fact, the modified gradient descent technique we use in Sec. 3.3 is the same as the iterative proportional fitting (also called iterative scaling algorithm) used to solve the parameters in exponential family graphical models.

It is important to note that because we use the slotted possible world semantics, we are able to factorize our partition function Z to a multi-linear polynomial raised to the power n (see Eq. 5). This simplification allows for drastic performance benefits in terms of solving and query answering Sec. 4.

9. RELATED WORK

Although there has been work in the theoretical aspects of probabilistic databases [43], as far as we could find, there is not existing work on using a probabilistic database for data summarization. However, there has been work by Markl [35] on using the maximum entropy principle to estimate the selectivity of predicates. This is similar to our approach except we are allowing for multiple predicates on an attribute and are using the results to estimate the likelihood of a tuple being in the result of a query rather than the likelihood of a tuple being in the database.

Although not aimed at data summarization, the work in [9] builds a probabilistic graphical model that is guaranteed to have efficient query answering for certain classes of queries, i.e. a tractable model. They propose a greedy search algorithm that simultaneously learns a Markov network and its underlying sentential decision diagram which gives a tractable representation of the network. While EntropyDB also learns a Markov network, our features are count queries over instances while in [9], they only use boolean valued features.

Our work is also similar to that by Suciu and Ré [42] except their goal was to estimate the size of the result of a query rather than tuple likelihood. Their method also relied on statistics on the number of distinct values of an attribute whereas our statistics are based on the selectivity of each value of an attribute.

Even though approximate query processing has been a major area of research in the database community for decades, there is still no widely accepted solution [32, 14, 36]. One main AQP technique is to use precomputed samples [3, 32, 14, 2]. In the work by Chaudhiri et al. [13], they precompute samples that minimize the errors due to variance in the data for a specific set of queries they predict. The work by [6] chooses multiple samples to use in query execution but only considers single column stratifications. VerdictDB [40] introduces variational subsamples as an alternative to bootstrapping and subsampling to provide efficient probabilistic guarantees. The core idea of variational subsampling is to loosen the restrictions on standard subsampling while still providing the same probabilistic guarantees. The work by [20] builds a measure-biased sample for each measure dimension to handle sum queries and uniform samples to handle count queries in order to provide a priori accuracy guarantees. Depending on if the query is highly selective or not, they choose an appropriate sample. Along a similar vein is the work of [33] which generates a unified synopses from a set of samples to answer queries approximately within a predefined error bound. The later work of BlinkDB [4] removes any assumptions on the queries. BlinkDB only assumes that there is a set of columns that are queried, but the values for these columns can be anything among the possible set of values. BlinkDB then computes samples for each possible value of the predicate column in an online fashion and chooses the single best sample to run when a user executes a query.

A main drawback for many systems relying on precomputed samples is that they require an existing workload to train on. Instead of using precomputed samples and needing a workload, the Quickr system in [29] injects sampling operators into query plans to generate samples on the fly for ad-hoc AQP in big data clusters. By using a variety of different samplers, they are able to handle distinct value queries as well as joins.

As EntropyDB does not use any samples nor does it require any existing workload, our approach is more closely related to the non-sample based AQP techniques of [12]. Much like EntropyDB answers queries directly on a probability distribution, the system in [12] builds multi-dimensional wavelets from a dataset and answers queries directly over the wavelet coefficient domain.

There are some AQP hybrid approaches such as the work in [45] which builds a deep learning model in order to draw better samples for AQP. They first build a collection variational autoencoders to learn the data distribution, and then, during run time, they use the model with rejection sampling to generate samples from the learned distribution. The system AQP++ [41] combines samples with precomputed aggregates, such as data cubes, to build a unified AQP system. For an input query, it uses samples to generate an estimate of the error between the true answer and the answer run on the closest precomputed aggregate and adds the error to the answer from the precomputed aggregate. IDEA [22] is a system that combines random sampling with indices targeted at rare populations to answer aggregate queries for interactive data exploration. The system also leverages results from previous aggregate queries and probabilistic query rewrite rules to approximately answer new queries using the old results, if possible.

The last main AQP technique is to use data sketches (e.g. count-min or KMV [7])[16, 32]. Sketches have the benefit of being able to handle streaming data but are usually built to handle a limited number of queries. For example, a KMV sketch answers how many distinct values there are but doesn't answer other aggregates.

Although not built for AQP specifically, the work in [48] uses Bayesian statistics and a few random samples to approximately answer what the extreme values are in the entire dataset. Using a historical query workload and Monte Carlo sampling, they generate a correction factor based on the shape of the query result and its associated error distribution. Then, they use the samples to generate a preliminary estimate and their learned correction factor to update their estimate. While this technique is probabilistic in nature, it is geared towards specifically answering extreme value queries rather than linear queries, like EntropyDB.

[39] and [30] both use deep learning networks to predict cardinalities for better query optimization. While EntropyDB can be used to estimate cardinalities for queries, we do not use deep networks and split our feature vector learning from optimization in that we decide which attributes to use for statistics before running our model learner. The deep network in [39] combines learning cardinalities with learning the best representation of a subquery.

10. CONCLUSION

We presented, EntropyDB, a new approach to generate probabilistic database summaries for interactive data exploration using the Principle of Maximum Entropy. Our approach is complementary to sampling. Unlike sampling, EntropyDB's summaries strive to be independent of user queries and capture correlations between multiple different attributes at the same time. Results from our prototype implementation on two real-world datasets up to 210 GB in size demonstrate that this approach is competitive with

sampling for queries over frequent items while outperforming sampling on queries over less common items.

11. REFERENCES

[1] http://www.transtats.bts.gov/.

[2] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *Acm Sigmod Record*, volume 29, pages 487–498. ACM, 2000.

[3] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *ACM Sigmod Record*, volume 28, pages 574–576. ACM, 1999.

[4] S. Agarwal et al. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proc. of EuroSys'13*, pages 29–42, 2013.

[5] D. A. Applegate, G. Calinescu, D. S. Johnson, H. Karloff, K. Ligett, and J. Wang. Compressing rectilinear pictures and minimizing access control lists. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1066–1075. Society for Industrial and Applied Mathematics, 2007.

[6] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 539–550, 2003.

[7] Z. Bar-Yossef, T. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer, 2002.

[8] M. Behrisch, B. Bach, N. Henry Riche, T. Schreck, and J.-D. Fekete. Matrix reordering methods for table and network visualization. In *Computer Graphics Forum*, volume 35, pages 693–716. Wiley Online Library, 2016.

[9] J. Bekker, J. Davis, A. Choi, A. Darwiche, and G. Van den Broeck. Tractable learning for complex probability queries. In *Advances in Neural Information Processing Systems*, pages 2242–2250, 2015.

[10] A. L. Berger, V. J. D. Pietra, and S. A. D. Pietra. A maximum entropy approach to natural language processing. *Comput. Linguist.*, 22(1), 1996.

[11] S. Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8, N0. 3-4:231–357, January 2015.

[12] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *The VLDB JournalThe International Journal on Very Large Data Bases*, 10(2-3):199–223, 2001.

[13] S. Chaudhuri, G. Das, and V. Narasayya. A robust, optimization-based approach for approximate answering of aggregate queries. In *ACM SIGMOD Record*, volume 30, pages 295–306, 2001.

[14] S. Chaudhuri, B. Ding, and S. Kandula. Approximate query processing: No silver bullet. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 511–519. ACM, 2017.

[15] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

[16] G. Cormode, M. Garofalakis, P. J. Haas, C. Jermaine, et al. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases*, 4(1–3):1–294, 2011.

[17] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. The case for interactive data exploration accelerators (ideas). In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, page 11. ACM, 2016.

[18] N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: diamonds in the dirt. *Communications of the ACM*, 52(7):86–94, 2009.

[19] A. Deshpande, M. N. Garofalakis, and R. Rastogi. Independence is good: Dependency-based histogram synopses for high-dimensional data. In *SIGMOD Conference*, 2001.

[20] B. Ding, S. Huang, S. Chaudhuri, K. Chakrabarti, and C. Wang. Sample+seek: Approximating aggregates with distribution precision guarantee. In *Proc. SIGMOD*, pages 679–694, 2016.

[21] C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, and A. Roth. Generalization in adaptive data analysis and holdout reuse. In *Advances in Neural Information Processing Systems*, pages 2350–2358, 2015.

[22] A. Galakatos, A. Crotty, E. Zgraggen, C. Binnig, and T. Kraska. Revisiting reuse for approximate query processing. *Proceedings of the VLDB Endowment*, 10(10):1142–1153, 2017.

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

[24] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *ACM SIGMOD Record*, volume 26, pages 171–182. ACM, 1997.

[25] A. Hosangadi, F. Fallah, and R. Kastner. Factoring and eliminating common subexpressions in polynomial expressions. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, 2004.

[26] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable approximate query processing with the dbo engine. *ACM Transactions on Database Systems (TODS)*, 33(4):23, 2008.

[27] P. Jetley et al. Massively parallel cosmological simulations with ChaNGa. In *Proc. IPDPS*, 2008.

[28] M. Jordan. An introduction to probabilistic graphical models. http://www.cs.cmu.edu/~lebanon/pub/book/, 2003.

[29] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri, and B. Ding. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *Proceedings of the 2016 international conference on management of data*, pages 631–646. ACM, 2016.

[30] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677*, 2018.

[31] C. Li et al. Optimizing linear counting queries under differential privacy. In *Proc. of PODS*, pages 123–134, 2010.

[32] K. Li and G. Li. Approximate query processing: what is new and where to go? *Data Science and Engineering*, 3(4):379–397, 2018.

[33] K. Li, Y. Zhang, G. Li, W. Tao, and Y. Yan. Bounded approximate query processing. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[34] E. Mäkinen and H. Siirtola. Reordering the reorderable matrix as an algorithmic problem. In *International Conference on Theory and Application of Diagrams*, pages 453–468. Springer, 2000.

[35] V. Markl et al. Consistently estimating the selectivity of conjuncts of predicates. In *Proc. of VLDB*, pages 373–384. VLDB Endowment, 2005.

[36] B. Mozafari and N. Niu. A handbook for building an approximate query engine. *IEEE Data Eng. Bull.*, 38(3):3–29, 2015.

[37] K. Murphy. Undirected graphical models. https://www.cs.ubc.ca/~murphyk/Teaching/CS340-Fall06/reading/ugm.pdf, 2006.

[38] L. Orr, M. Balazinska, and D. Suciu. Probabilistic database summarization for interactive data exploration. *Proceedings of the VLDB Endowment*, 10(10):1154–1165, 2017.

[39] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi. Learning state representations for query optimization with deep reinforcement learning. *arXiv preprint arXiv:1803.08604*, 2018.

[40] Y. Park, B. Mozafari, J. Sorenson, and J. Wang. Verdictdb: universalizing approximate query processing. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1461–1476. ACM, 2018.

[41] J. Peng, D. Zhang, J. Wang, and J. Pei. Aqp++: connecting approximate query processing with aggregate precomputation for interactive analytics. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1477–1492. ACM, 2018.

[42] C. Ré and D. Suciu. Understanding cardinality estimation using entropy maximization. *ACM TODS*, 37(1):6, 2012.

[43] D. Suciu, D. Olteanu, C. Ré, and C. Koch. Probabilistic databases. *Synthesis Lectures on Data Management*, 3(2):1–180, 2011.

[44] Y. W. Teh and M. Welling. On improving the efficiency of the iterative proportional fitting procedure. In *AIStats*, 2003.

[45] S. Thirumuruganathan, S. Hasan, N. Koudas, and G. Das. Approximate query processing using deep generative models. *arXiv preprint arXiv:1903.10000*, 2019.

[46] K. Tzoumas, A. Deshpande, and C. S. Jensen. Efficiently adapting graphical models for selectivity estimation. *The VLDB Journal*, 22(1):3–27, 2013.

[47] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1-2), Jan 2008.

[48] M. Wu and C. Jermaine. A bayesian method for guessing the extreme values in a data set? In *Proceedings of the 33rd international conference on Very large data bases*, pages 471–482. VLDB Endowment, 2007.

[49] E. Yang, P. Ravikumar, G. I. Allen, and Z. Liu. Graphical models via univariate exponential family distributions. *The Journal of Machine Learning Research*, 16(1):3813–3847, 2015.