Building and Querying Probabilistic Models for Open World Database Systems

Laurel Orr

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2019

Reading Committee:

Professor Magda Balazinska, Chair

Professor Dan Suciu, Chair

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Building and Querying Probabilistic Models for Open World Database Systems

Laurel Orr

Co-Chairs of the Supervisory Committee:
Professor Magda Balazinska
The Paul G. Allen School for Computer Science and Engineering

Professor Dan Suciu
The Paul G. Allen School for Computer Science and Engineering

A fundamental assumption of traditional database management systems is that the database contains all information necessary to answer a query; i.e., the database contains the entire population of data. However, with the increasing availability of public data samples (e.g., government data) and easy-to-use scientific programming languages (e.g., Python), data scientists are turning to samples to analyze and understand the population they represent. As databases do not treat stored data as samples, data scientists are forced to use tools outside of the database for their data processing needs.

For database management systems to accommodate this growing group of users, they need to adopt the open world assumption that tuples not in the database still exist. In this dissertation, we answer two main research questions on how to build an open world database system that approximately answers queries as if they were issued over the entire population.

The first question is: in an ideal setting where we can choose what statistics to gather about a population, how can we build a probabilistic model of the population that assumes all tuples have some nonzero probability of existing, i.e., the open world assumption. By using the Principle of Maximum Entropy, we built a prototype database system called EntropyDB that builds a probabilistic model for approximate query processing.

The second question is: when the database just has access to a sample of the population and some population aggregate information, how can we automatically remove arbitrary selection bias to allow users to accurately answer population queries. We implement this automatic debiasing in THEMIS, the first open world database system that uses a priori population aggregate information to rebalance sample data.

While there is still important future work in the field of open world database system, this dissertation presents the first step towards its realization.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

To all the negative five people who are actually going to read the acknowledgments of a computer science 150-plus page thesis, I know what you expect to see in this section. But this is the one section where I do not have to be formal or proper or "technical". So, you are getting the raw Laurel acknowledgments.

In standard acknowledgment sections, there is the thanking of the advisors, for whom I could not have done this dissertation, and then the thanking of the parents and the friends and the other life supporters. All this is true. I could not have done this without my advisors, Dan and Magda. Their unending optimism and support made this possible. And of course, my friends and lab mates who somehow managed to put up with me for six years and kept me sane and happy. To roommate of 6 years who is also getting a PhD in Economics, I seriously would not have survived without you. Not only were you the only voice of reason in my head (telling me to sleep when sick and important stuff like that), but the hours you spent listening to my inner ramblings and helping me process was an incredible gift. (Yes, that's cheesy. I know.) My parents literally made me. So, that's important. (I love you Mom and Dad!).

You know what. Instead of me iterating what most acknowledgment sections say[1], below is a word cloud of some recent UW Computer Science dissertation acknowledgments sections. It pretty much says it all. Well, the little "ph" does not mean much to me (from the word cloud generator separating Ph.D. into Ph and D), and I enjoy how Amazon's Alexa is in there, too. Amazon gives us money, which we appreciate.

But let's get real about what has led me to be here today. I want to pause and first

[1]While typing this, I have not spelled acknowledgment correctly once without the use of spell check.

Figure 1: Wordcloud of a sample of recent University of Washington Computer Science acknowledgment sections.

thank my parents' cats. Yes, cats. These three cats have not only terrorized my parents' lives (it's a long story), but they also have helped motivate me. To Sebastian, the most loved cat who has shown me that being the fluffiest certainly helps in life. To Thea, who is the devil incarnate[2]. She reminds me of how bad my life would be if some demonic being lived with me instead of my parents. (I love you Mom and Dad!). That somehow made me feel better in hard times. And to Jeremy, who is literally the dumbest creature to live on this earth[3]. He makes me so thankful that I am not that dense. I do not think that bushes are solid and can always find the open door in front of me.

These cats, however, are more recent additions to my motivation in life. I think we need to go further back, back to my introduction to computers. Most computer scientists talk about tinkering with computers and finding it amazing or learning to code at an early age. My earliest memory: sitting with my dad playing Doom[4]. Mind you, I was six, six and yelling at the computer "Kill it! Kill it! Get the shotgun out! KILL!" To say the least, my

[2]After showing this comment to my sister, she claims Thea is not the devil. Thea just knowns what she deserves

[3]After showing this to my Dad, he claims Jeremy has become immune to criticism. Jeremy calls is Fake News

[4]For those unfamiliar with Doom, it's a bloody, violent game where you run around dungeons killing monsters with a variety of death-inflicting weapons. Most importantly, as your health decreases, the face of your character gets steadily more and more covered with blood.

mom was not happy when she found out.

Thinking back on this, maybe it was not the deciding factor in me becoming a computer scientist. Had I become an assassin, then yes, that experience would probably have played a role. But as I am a not-very-sneaky person, I don't think an assassin would have been a good career for me[5]. There was another time when I was also six and decided to bang my forehead against a counter just to "see what happens". Look, the scientist in me was already coming out. Suffice it to say, blood came pouring down my face, and I still have a scar on my forehead. The connection between bloody Doom man and my bleeding forehead is not lost on me. Good job parents. (I love you Mom and Dad!).

Right, back to my dissertation. (I told you this was an abnormal acknowledgment section). I owe a lot to one of my best friends who either got me into or got me out of a lot of trouble when I was younger. She and I were the two who decided it would be a really good idea to eat about two pounds of jelly beans over the span of one Easter day. We had a container of 49 flavors of jelly beans and had to try every possible combination. See, even then I was learning about brute force. I just remember curling up on the floor wishing for death by the evening. One of my finer moments in life.

I owe a lot of my perseverance to my mom, who would have me untangle bundles of antique jewelry when I was younger. For me, it was an excuse to watch some television, and I would untangle for hours until my fingers were silver with tarnish.

My dad, besides teaching me how to kill monsters, was the family funny man. However, I will never forgive him for causing us to lose the father-daughter swimming race on a Fourth of July when I was around six (I seem to remember a lot of when I was six, and then I remember like nothing until yesterday.). I swam my little heart out to tag him in, and the minute he gets into the water, he stops. After what feels like forever, he finally starts up again, but by then, we have already lost. I was furious. Even though I now know he stopped

[5]The ACT career recommendation actually said I should be a brick layer. I didn't even know that was a career.

swimming because his swim shorts had fallen off, and he decided putting them back on was the optimal choice, I don't know if I can forgive him. What have I learned from him? Duct tape your shorts on.

I should also thank my brother and sister for their support. Although I had to lock away my make-up from my sister when we were in high school, she has been a role model for me. To my brother, there is no one else I would rather play hours of video games with. You let me get away with secretly stealing all your treasure. And I should thank my nephew, who reminds me how nice it is to be able to reach the upper cabinets in the kitchen. Although, I am very jealous that he gets to sleep like 11 hours a night.

Let's face it. Getting a PhD requires amazing people to support you but also a lot of happy circumstances (e.g., landing on the right project idea, meeting the right people, etc.) and intense stubbornness. I've had all of those in spades. So, to everyone who I have ever thanked, I thank you again. To all those I haven't thanked, why not thank you, too. (I love you Mom and Dad!).

Chapter 1

INTRODUCTION

In fields ranging from sports to economics, discovery and decision making are being increasingly driven by the analysis of large, diverse datasets. This shift encourages scientists to continue to push the boundaries of what can be learned from data.

As the amount of data increases, data scientists need tools and systems to help them store and analyze their data. Database management systems (DBMSs) should be the ideal solution since they help users manage and process their data. However, a fundamental issue prevents data scientists from taking advantage of the speed, structure, and ease a database provides: data scientists analyze data samples, while databases answer queries as if they can access the entire data population.

For example, suppose a data scientist wants to use a sample of flights landing in Florida to pose queries about all flights in the United States. If the scientist ingests this sample dataset into a database, the database will treat the data as if it represents all flights in the United States. This assumption can lead to inaccurate analytical results. For example, database queries will indicate that all flights in the United States land in Florida.

Even if the data scientist avoids asking about flight destinations, which is clearly inaccurately represented in the sample, subtle biases in the sample will still lead to incorrect results. For example, flights landing in Florida tend to be, on average, longer than flights landing in other states. Scientists asking about flight time will consequently get incorrect overestimates.

This difference between a sample and an entire population is problematic because samples are incomplete parts of complete information that can be arbitrarily biased. Without correcting for this bias, results learned from sample analytics can be inaccurate. Database

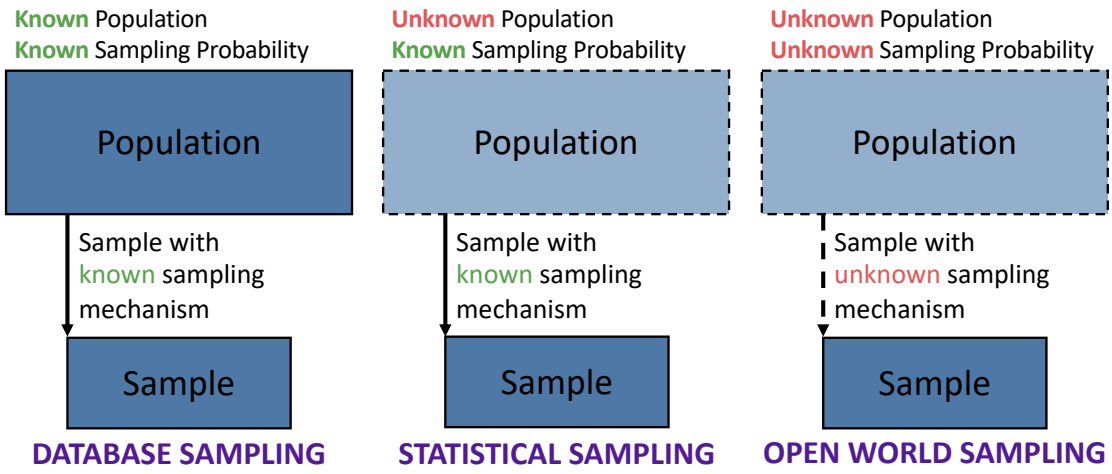


Figure 1.1: Three fundamental types of sampling.

systems process the data as if it were complete and do not correct for sample bias. Thus, the task of debiasing data is left for users to manually perform. A database meant to help users analyze data cannot help data scientists analyze their samples.

In this dissertation, we address the incompatibility between database data and sample data by researching *open world database systems*, which assume that stored data is incomplete. We design and prototype two different open world database systems that automatically correct for sample bias and help users pose questions over the population they seek to study. The first system, EntropyDB, takes a probabilistic approach to population query answering by learning a probability distribution over the population. The second system, Themis, merges a probabilistic technique and a sample rebalancing technique to answer queries. Both systems take a critical first step towards building an open world database system.

We begin by introducing three high-level types of sampling (Figure 1.1) that motivate sample analysis and explain the necessity of open world database systems. We then introduce the two open world database systems developed in this thesis and layout the dissertation's structure.

1.1 Statistical Sampling

In the paper *The Rise of Survey Sampling* [27], the author posits that using samples to examine some population is "as old as mankind." Over the last century, sampling has become a widely accepted statistical method used by data scientists across numerous domains. At its core, sampling lets scientists quickly analyze and pose questions about some population using a representative subset of the data.

Building a sample requires information about both the *population* and *how individuals are selected*. The latter is also referred to as the *sampling mechanism* (sampling inclusion probability or propensity score [18, 27, 88]). An example population could be every individual living in the United States, or every graduate student enrolled in a Computer Science program. Example sampling mechanisms are *uniform random sampling* (all individuals are sampled with equal probability) or *stratified sampling* (each predefined subgroup is randomly sampled).

Traditional sampling, i.e., *statistical sampling*, arises when scientists lack access to the entire population of data they want to investigate but do know the sampling mechanism. For example, take the population of all individuals living in the United States. To preserve privacy, this dataset in its entirety is not publicly available; however, the US Census Bureau releases samples to the public through The American Community Survey Public Use Microdata Sample (PUMS) [7]. These samples include information on the inclusion probability of each individual, meaning the sampling mechanism is known to scientists who want to analyze this dataset.

Now consider the alternative example population of all Computer Science graduate students in the United States. This dataset does not exist, nor are samples publicly available. However, a data scientist could design a sampling procedure to select random graduate students from random universities. As the scientist designs the sample, the probability of certain graduate students being included in the sample becomes known.

Knowing the sampling mechanism is critical for accurate sample analysis because the

sample may contain bias that without statistical correction would make it unrepresentative of the population. For example, suppose a data scientist has a sample of United States population data that includes the commute time to work for each individual. The scientist wants to know the average commute time across the United States. Further, suppose the sample is biased towards individuals in New York, meaning NY residents are overrepresented in the sample. If the data scientist calculates the average commute time of this sample and reports it as the average commute time of the United States, the result will be an overestimate because individuals living in NY have, on average, a longer commute than individuals in other states [11].

By knowing the sampling mechanism, either by having a priori knowledge of the sampling probability or by designing and building the sample, data scientists can use common techniques like sample reweighting (weighting the points in a sample to better represent the unbiased distribution) to accurately analyze the sample [88]. These techniques correct for bias in the sample. In the NY commute time example, sample reweighting would assign a smaller weight to individuals living in NY than to those living in other states. Using this weight to compute a weighted average commute time lets data scientists receive more accurate results.

1.2 Database Sampling

Statistical sampling is not the only tool invented in the last century to help data scientists analyze data. Database management systems (DBMSs), invented in the 1960's, help users store, manage, and analyze their data. They specialize in letting users not only store entire populations of data but also ask questions about this data using a straightforward declarative query language. This language lets users declare what data they want without worrying about how to retrieve it.

It is important to highlight a fundamental assumption of database systems, called the *closed world assumption*. This assumption states that the entire population of data is contained in the database. Any information not in the database is assumed to not exist. This

assumption has led database systems to focus on answering queries efficiently because, as long as the queries are answered using all the data, they are completely accurate.

Efficient query processing has become a challenge, however, as datasets enlarge. For example, astronomers who analyze telescope images from sky surveys like LSST must manage approximately twenty terabytes of data per night [9]. Processing this massive amount of data efficiently is non-trivial, even for DBMSs. To support these big data analysts, databases have been forced to adapt and evolve with a variety of new systems and techniques.

One such solution has been to utilize sampling for *approximate query processing* (AQP). AQP systems answer population queries more efficiently than if they were run on the entire data population in exchange for adding some error to the query result. In other words, they answer population queries approximately but quickly. Despite the variety of AQP systems [87, 37, 121, 96], the gold standard AQP technique is *database sampling*, also known as closed world sampling.

For example, BlinkDB [16] uses a historical workload of queries to optimally select which samples to precompute. At runtime, it selects a sample based on the user's acceptable error margin or maximum runtime. Sample+Seek [50] develops a measure-biased sampling scheme to guarantee that answers to aggregate queries meet a user-specified error bound; it further utilizes specialized index structures to answer queries over values not contained in the sample. The AQUA system [15, 60] uses precomputed join samples and stratified samples to answer queries.

Hence, database or closed world sampling is simply sampling from the database where the population is known. These AQP systems design a sampling mechanism and then create samples by accessing the entire population. Contrast this to statistical sampling, where the entire population cannot be accessed. However, as both types of sampling know (or create) the sampling mechanism, samples can be analyzed using existing debiasing techniques.

1.3 Open World Sampling

While the closed world assumption is ideal for DBMSs and AQP, it is ill-suited to handle a relatively new data source: samples from data repositories. Over the past decade, online data repositories have become more popular and prevalent. For example, the website Data.World [8], started in 2015, provides users with access to thousands of samples ranging from the popularity of cats versus dogs in the US to public parking counts in New York. Alternatively, social media websites like Twitter provide public APIs to download sample social media data [12]. These repositories provide quick and easy access to population samples.

The challenge with using these samples is that they are *open world samples*, meaning both the underlying population and the sampling mechanism are not known. Hence, the samples contain arbitrary bias that cannot be corrected using standard statistical methods. In the NY commute example, if the data scientist does not know that individuals in NY are overrepresented, selection bias cannot be corrected.

Existing research on techniques for open world sample analysis fall into three high-level categories. The first is to measure only quantities of interest that can filter out or remove sample bias. For example, in [133], the authors explain how measuring a "difference in differences" between quantities of interest can cancel out the bias. However, they caution that no general approach to debiasing works, and each model and measurement must be carefully examined to ensure accuracy.

The second and third techniques rely on another increasingly available data resource: population aggregates (tabulations) [97]. Along with the increase in the number of publicly available data samples, there has been a recent push for more data transparency and reporting by corporations and governments, e.g., the United State's OPEN Government Data Act passed in 2018 [1] and the InFuse UK aggregate population statistics tool [2]. These reports are often in the form of population aggregate queries. For example, the FBI's 2017 Internet Crime Report [3] presents a table showing an aggregate query over crime type, counting the number of victims in each crime type group.

The second technique, which uses these tabulation, focuses on learning the population directly and ignoring any biased sample (called *synthetic reconstruction* in demography literature [72]). This method also arises when there is no biased sample to use (i.e., only population aggregates exist). As these aggregates give partial knowledge of the probability of various combinations of attributes, they can be used to probabilistically *generate* a synthetic representative population sample [72, 113, 54].

The third technique treats these aggregates as constraints when modeling some population quantity of interest or modeling sample bias [89, 120]. For example, in [134], the authors manually add a bias factor when measuring the number of births for some geographical locations and demographic groups from biased sample data. They use population aggregates to learn this bias factor. Alternatively, in [89], the authors use an iterative reweighting algorithm to reweight a sample of individuals living in the UK with geographical aggregates over UK administrative units.

All these techniques, however, suffer from one main bottleneck to analysis: they require manual, error-prone solutions specialized for a specific problem and dataset. There is no general, standardized system or solution that automatically performs data debiasing on arbitrary samples and answers population queries.

1.4 Thesis Contributions

The fundamental problem addressed in this thesis is as follows. **Data scientists wanting to analyze arbitrarily biased samples without knowing the sampling mechanism cannot utilize databases or existing research in database sampling. Databases make the closed world assumption, which directly contradicts the open world nature of these samples. For databases to address the needs of this growing user base, they need to make an open world assumption and automatically correct for sample bias.**

We posit that data debiasing and approximate population query answering can be solved by an open world database system that automatically corrects for sample bias and answers

queries approximately as if they were asked over the entire population. No such system yet exists because the open world assumption is at odds with the closed world assumption currently adopted by DBMSs.

Building an open world database system, however, is challenging since all existing general purpose sample debiasing techniques require knowledge of the sampling mechanism. While theoretical work describes a probabilistic open world database system where tuples represent facts that have a probability of being true [33, 62], no existing research addresses open world database systems for debiasing samples.

In this dissertation, we research and build two prototype open world database systems for approximate population query processing where each system answers a different research question with respect to open world population analysis. These research questions take an important first step towards building a fully realized open world database system.

Our first research question is: assuming you can optimally choose the population aggregates, how can you build a probabilistic model of the population for approximate query processing? We answer this question by building EntropyDB, the first database system that builds a probabilistic model of a population dataset based on aggregates using the Principle of Maximum Entropy (see below). We explore how to build such a model efficiently with novel compression techniques and how to answer queries at interactive speeds. This system was published in VLDB 2017 [104], and an extended version is under review for VLDBJ 2019.

Our second research question is: given a biased sample and some population aggregates, how can you automatically debias the sample? We answer this question by building Themis, an open world database system that combines sample reweighting and probabilistic modeling techniques to answer queries approximately. We consider Themis to be the first open world database for automatic data debiasing.

Woven through our two systems is one unifying mathematical principle for open world analysis: the Principle of Maximum Entropy (MaxEnt). The MaxEnt principle states that the optimal probability distribution is the one with maximum entropy and consistent with

the current state of knowledge. This principle is omnipresent both in theoretical domains, such as mathematics and computer science, and in practical domains, such as image reconstruction, taxation policies, and species distribution modeling [69, 108]. The MaxEnt principle is a natural fit for open world analysis since it learns the best choice population probability distribution subject to known, testable information [30]. In open world analysis, the biased sample gives testable information that is known to be true. This information acts as constraints when learning the best true population distribution, making the principle ideal for an open world database.

We now describe each system in more detail.

1.4.1 EntropyDB*: A Probabilistic Approach to Approximate Query Processing*

In Chapter 2 we address how to build an AQP system, called EntropyDB, that uses population aggregate information to learn a probability distribution of the population. While we frame this work as an AQP system, at its core it is an open world database system because it uses only population aggregates to model the population and answer queries.

We address three main challenges in this work. The first is *how to build a probabilistic model given aggregates*. To do this we adopt the possible world semantics and use MaxEnt to learn a distribution subject to constraints learned from our given aggregates. This enforces that the learned distribution must fit the true distribution.

The next challenge is *how to learn this distribution efficiently*. To make solving tractable, we design a novel factorization technique that compresses our learned distribution and improves performance. We also develop a factorization algorithm that efficiently computes this compression.

The last challenge is *how to answer queries efficiently*. On top of our compression technique, we develop an algorithm for answering queries that involves taking the derivative of the mathematical representation of our distribution. We further optimize this derivation by showing how it can be replaced by setting certain terms in our mathematical representation to zero.

We conclude by experimentally showing how EntropyDB is comparable to state-of-the-art AQP techniques in terms of accuracy and is more accurate than these techniques at indicating whether tuples exist in the population.

1.4.2 Themis*: Sample Debiasing in an Open World Database System*

In Chapter 3 we answer how to build an open world database system using a biased sample and incomplete aggregates, meaning the aggregates may not contain all attributes present in the sample. The system we build, called Themis, merges two fundamental techniques to answer population queries approximately: sample reweighting and population probabilistic modeling.

For *sample reweighting*, we rebalance the sample by assuming each tuple has some associated weight that indicates the number of tuples in the population it represents. We first assume this weight is a linear combination of a tuple's attributes and use linear regression to learn the parameters of this linear function. We then compare this linear assumption to assuming the weight is exponentially related to a tuple's attributes. We learn the parameters of this non-linear weight function using an existing technique called Iterative Proportional Fitting (IPF) [89].

For *probabilistic modeling*, we use Bayesian networks to learn the population probability distribution. We develop novel algorithms to learn the Bayesian network's structure and parameters since traditional Bayesian network learning algorithms assume access to the entire population. We have access only to a biased sample and population aggregates.

Finally, we develop a simple heuristic for choosing which of these two methods to use during query answering. We experimentally show how Themis achieves better accuracy than existing sample debiasing technique and is robust to samples that do not have the same support as the population, meaning that some tuples in the population will never be sampled (i.e., their sampling probability is zero).

In summary, this thesis addresses how to use population aggregate information and biased samples to build an open world probabilistic database. This database automatically debiases

samples and lets users get approximate answers to population queries.

The rest of this dissertation is organized as follows. We first explore EntropyDB (Chapter 2) and Themis (Chapter 3). Then, we review related work in Chapter 4 and conclude and discuss possible directions for future work in Chapter 5.

Chapter 2

ENTROPYDB: A PROBABILISTIC APPROACH TO APPROXIMATE QUERY PROCESSING

As mentioned in Chapter 1, the research question addressed in this chapter is how to build a probabilistic model of a population using an optimal set of aggregates. In other words, we assume we have access to the population in order to choose our aggregates before using them to build a probabilistic model. Although we frame this problem as one of approximate query processing and interactive data exploration (defined below), we are performing population synthesis from optimal aggregates.

Interactive data exploration allows a data analyst to browse, query, transform, and visualize data at "human speed" [43]. It has been long recognized that general-purpose DBMSs are ill suited for interactive exploration [96]. 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 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* [67] or systems like DBO [75] and Quickr [78], or precomputed offline, like in BlinkDB [16] or Sample+Seek [50]. 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 chapter, 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 [65] is a MaxEnt model for both differentially private and, by [53], 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 chapter, 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 (Section 2.2). We show how to simplify the MaxEnt model to be a multi-linear polynomial that has one monomial for each possible tuple (Section 2.2, Equation 2.6) rather than its naïve form that has one monomial for each possible world (Section 2.1, Equation 2.2). Even with this simplification, the MaxEnt model starts by being larger than the data. For example, our smaller experimental dataset (introduced in Section 2.6) 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 (Section 2.3.1); 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 (Section 2.3.2) 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 [94] 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 [124] of a *dual* function (Section 2.1), 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 [29], 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 chapter, we develop the following new techniques:

- 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 (Section 2.2).
- A compression technique and optimized implementation of the compression for the MaxEnt summary (Section 2.3.1, Section 2.4.1).
- Optimized query processing techniques, including implementation details (Section 2.3.2,

Section 2.4.2).

- A new method for selecting 2-dimensional statistics based on optimal matrix reorderings and a modified K-D tree (Section 2.5)
- Detailed experiments comparing the accuracy of the MaxEnt summary versus various sampling techniques (Section 2.6.3).
- Solving time and query runtime evaluations showing our interactive query speeds (Section 2.6.4, Figure 2.6.4).
- A discussion on how the MaxEnt summary relates to probabilistic databases and graphical models (Section 2.7).
- A description on how the MaxEnt summary can be extended to handle data updates and joins (Section 2.7.2).

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

2.1 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.

2.1.1 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[1]. This approach captures the idea that the distribution should be uniform except where otherwise specified by the given constraints.

2.1.2 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, PWD, the probability distribution

[1]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.

$\Pr(I)$ 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:

$$s_j = \mathbb{E}[\phi_j(I)], \quad j = 1, k \tag{2.1}$$

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 [26, 124, 111], the MaxEnt probability distribution takes the form

$$\Pr(I) = \frac{1}{Z} \exp\left(\sum_{j=1}^{k} \theta_j \phi_j(I)\right) \tag{2.2}$$

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 non-linear system of k equations, Equation 2.1, which is computationally difficult. However, it turns out [124] that Equation 2.1 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

m	# attrs
k	# statistics
D_i	domain of A_i
N_i	$\lvert D_i \rvert$
$\mathbf{q}$	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 2.1: EntropyDB common notation.

the *cumulant*.

Lastly, we adopt a slightly different notation where instead of e^{θ}, we use α. Equation 2.2 now becomes

$$\Pr(I) = \frac{1}{Z} \prod_{j=1}^{k} \alpha_j^{\phi_j(I)}. \tag{2.3}$$

2.2 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 2.1 lists common symbols and their definitions, and Table 2.2 lists various assumptions we incrementally make on our model and the section they are introduced.

Queries are limited to linear queries.	Section 2.2
Continuous attributes are discretized.	Section 2.2
Summary includes all 1-dimensional statistics.	Section 2.2
Statistics are collections of range predicates.	Section 2.3.1
Each set of 2D statistics is disjoint.	Section 2.3.1
Summary adds only 2D high order statistics.	Section 2.5.1

Table 2.2: EntropyDB model assumptions, and the section they are introduced.

2.2.1 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.[2] Let $Tup = D_1 \times D_2 \times \cdots \times D_m = \{t_1, \ldots, t_d\}$ be the 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[3] $\mathbf{n}^I = [n_1^I, \ldots, n_d^I] \in \mathbb{R}^d$, where each number n_i^I represents the count of the tuple $t_i \in Tup$ in I (Figure 2.1). 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 chapter, we limit our main analysis to linear queries but do discuss how to handle joins in Section 2.7.2. 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, $\langle \mathbf{q}, \mathbf{I} \rangle = \sum_i n_i^I \langle \mathbf{q}, \mathbf{t}_i \rangle$.

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

[3] This is a standard data model in several applications, such as differential privacy [85].

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:

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

Query:

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

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

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

Figure 2.1: Illustration of the data and query model

Figure 2.1 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 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 Equation 2.3 becomes

$$\Pr(I) = \frac{1}{Z} \prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{I} \rangle}. \tag{2.4}$$

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

Lemma 2.2.1. *The partition function is given by*

$$Z = P^n \tag{2.5}$$

where P is the multi-linear polynomial

$$P(\alpha_1, \ldots, \alpha_k) \stackrel{def}{=} \sum_{i=1,d} \prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{t}_i \rangle}. \tag{2.6}$$

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,

$$\begin{aligned} Z &= \sum_I \Pr(I) = \sum_{\mathbf{n}:||\mathbf{n}||_1 = n} \frac{n!}{\prod_i n_i!} \prod_{j=1,k} \alpha_j^{\sum_i n_i \langle \mathbf{c}_j, \mathbf{t}_i \rangle} \\ &= \left(\sum_{i=1,d} \prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{t}_i \rangle} \right)^n = P^n. \end{aligned}$$

□

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 (Equation 2.6), 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 2.2.2. *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[4], and γ_1, γ_2 are for C.

Consider the concrete instance that satisfies the above statistics

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

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 2.2.3. *Continuing the previous example, we add the following multi-dimensional*

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

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

$$\begin{aligned} P = &\alpha_1\beta_1\gamma_1[\alpha\beta]_{1,1}[\beta\gamma]_{1,1} + \alpha_1\beta_1\gamma_2[\alpha\beta]_{1,1} + \\ &\alpha_1\beta_2\gamma_1[\beta\gamma]_{2,1} + \alpha_1\beta_2\gamma_2 + \\ &\alpha_2\beta_1\gamma_1[\beta\gamma]_{1,1} + \alpha_2\beta_1\gamma_2 + \\ &\alpha_2\beta_2\gamma_1[\alpha\beta]_{2,2}[\beta\gamma]_{2,1} + \alpha_2\beta_2\gamma_2[\alpha\beta]_{2,2}. \end{aligned} \tag{2.7}$$

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 Equation 2.6.
- 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 $A_2 \in [20, 40] \wedge A_4 = 20$.

- 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* [124, 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

$$P = \sum_{j \in J_i} \alpha_j P_j \tag{2.8}$$

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 2.2.3, the 1-dimensional variables for A are α_1, α_2, and indeed, each monomial in Equation 2.7 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 Equation 2.7. P is also linear in β_1, β_2 and in γ_1, γ_2.

2.2.2 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 $\Pr(I)$ (Equation 2.4 and 2.6). 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

$$\mathbb{E}[\langle \mathbf{c}_\ell, \mathbf{I} \rangle] = \frac{n\alpha_\ell}{P} \frac{\partial P}{\partial \alpha_\ell}. \tag{2.9}$$

We review the proof here. The expected value of $\langle \mathbf{c}_\ell, \mathbf{I} \rangle$ over the probability space (Equation 2.4) is

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

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:

$$P_\mathbf{q}(\alpha_1, \ldots, \alpha_k, \beta) \stackrel{\text{def}}{=} \sum_{i=1,d} \prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{t}_i \rangle} \beta^{\langle \mathbf{q}, \mathbf{t}_i \rangle} \tag{2.10}$$

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 Equation 2.9 to the query $\mathbf{q}$ to derive:

$$\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle] = \frac{n}{P} \frac{\partial P_\mathbf{q}}{\partial \beta}. \tag{2.11}$$

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 Equation 2.11. One way to obtain $P_\mathbf{q}$ is to iterate over all monomials in P and add β to the monomials corresponding to tuples counted by $\mathbf{q}$. As this iteration is inefficient, Section 2.3.2 describes how to avoid modifying the polynomial altogether.

2.2.3 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 Section 2.1, this is equivalent to maximizing the dual function Ψ:

$$\Psi \stackrel{\text{def}}{=} \sum_{j=1}^{k} s_j \ln(\alpha_j) - n \ln P. \tag{2.12}$$

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 Equation 2.9. 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 [29], 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 $P_{\alpha_j} \stackrel{\text{def}}{=} \frac{\partial P}{\partial \alpha_j}$ and solving, we obtain:

$$\alpha_j = \frac{s_j(P - \alpha_j P_{\alpha_j})}{(n - s_j)P_{\alpha_j}}. \tag{2.13}$$

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.

2.3 Logical Optimizations

We now discuss two logical optimizations: (1) summary compression in Sec. 2.3.1 and (2) optimized query processing in Sec. 2.3.2. In Section 2.4, we discuss the implementation of these optimizations.

Algorithm 1 EntropyDB 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 − nα_j P_{α_j} / P
      maxError = max(error, maxError)
```

2.3.1 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 2.2.3.

Example 2.3.1. *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*

$$\begin{aligned} P =& (\alpha_1+\alpha_2)(\beta_1+\beta_2)(\gamma_1+\gamma_2)+ \\ & (\gamma_1+\gamma_2)(\alpha_1\beta_1([\alpha\beta]_{1,1}-1)+\alpha_2\beta_2([\alpha\beta]_{2,2}-1))+ \\ & (\alpha_1+\alpha_2)(\beta_1\gamma_1([\beta\gamma]_{1,1}-1)+\beta_2\gamma_1([\beta\gamma]_{2,1}-1))+ \\ & \alpha_1\beta_1\gamma_1([\alpha\beta]_{1,1}-1)([\beta\gamma]_{1,1}-1)+ \\ & \alpha_2\beta_2\gamma_1([\alpha\beta]_{2,2}-1)([\beta\gamma]_{2,1}-1). \end{aligned} \tag{2.14}$$

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 Theorem 2.3.3.

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 (Figure 2.3 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 Figure 2.2 (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 [70]; 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 tech-

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

Figure 2.2: Example of a compressed polynomial P after applying the inclusion/exclusion principle.

niques infeasible. We leave it as future work to investigate other factorization techniques geared towards massive polynomials.

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

- 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 $\rho_{iS} \stackrel{\text{def}}{=} \bigwedge_{j \in S} \rho_{ij}$, and $\pi_S \stackrel{\text{def}}{=} \bigwedge_i \rho_{iS}$; 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 multi-dimensional 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 multi-dimensional statistics for the attribute set defined by $\mathcal{I}$. (These parameters are discussed further in Section 2.5).

Finally, define $J_{\mathcal{I}^+} \subseteq \mathcal{P}([k])$[5] for $\mathcal{I}^+ \subseteq \mathcal{P}([m])$ to be 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 $\texttt{false}$). In other words, for each $S \in J_{\mathcal{I}^+}$, $\rho_{iS} \notin \{\texttt{true}, \texttt{false}\}$ for $i \in \bigcup \mathcal{I}^+$ and $\rho_{iS} \equiv \texttt{true}$ 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 $J_{\{\{1,2,3\}\}} = \{\{j_4\}\}$.

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

$\{\{j_2, j_3\}\} \notin J_{\{\{2,3\}\}}$ because $\rho_{2j_2} \wedge \rho_{2j_3} \equiv \texttt{false}$. Further, $J_{\{\{1,2\},\{2,3\}\}} = \{\{j_1, j_2\}\}$ because $\rho_{2j_1} \wedge \rho_{2j_2} \not\equiv \texttt{false}$. However, $\{j_1, j_3\} \notin J_{\{\{1,2\},\{2,3\}\}}$ because $\rho_{2j_1} \wedge \rho_{2j_3} \equiv \texttt{false}$, 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 Theorem 2.3.2.

Theorem 2.3.2. *The polynomial P is equivalent to:*

$$P = \overbrace{\left(\prod_{i\in[m]} \sum_{j\in J_i} \alpha_j\right)}^{(i)} + \left[\sum_{\mathcal{I}\subseteq[m]} \overbrace{\left(\prod_{i\notin\mathcal{I}} \sum_{j\in J_i} \alpha_j\right)}^{(ii)} \overbrace{\sum_{\ell=1}^{B_a} \sum_{\substack{\mathcal{I}^+\subseteq\mathcal{P}([m]),\\ |\mathcal{I}^+|=\ell,\\ \bigcup\mathcal{I}^+=\mathcal{I}}} \sum_{S\in J_{\mathcal{I}^+}}}^{(iii)} \underbrace{\left(\prod_{i\in\bigcup\mathcal{I}^+} \sum_{\substack{j\in J_i,\\ \pi_j\wedge\rho_{iS}\not\equiv \texttt{false}}} \alpha_j\right)\left(\prod_{j\in S}(\alpha_j - 1)\right)}_{(iii)}\right]$$

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 $(\pi_j \wedge \rho_{iS}) \not\equiv \texttt{false}$ 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 Section 2.4.1. The algorithm can be used during query answering to compute the compressed representation of $P_{\mathbf{q}}$ from P (Sec. 2.2.2) 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 Section 2.3.2.

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 2.3.3. *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 $O(\prod_{i=1}^m N_i)$ which is approximates 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 $O(\prod_{i=1}^m N_i)$.

2.3.2 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

$$\pi = \rho_1 \wedge \cdots \wedge \rho_m \tag{2.15}$$

and ρ_i is a predicate over the attribute A_i. If $\mathbf{q}$ ignores A_i, then we simply set $\rho_i \equiv \texttt{true}$. Our goal is to compute $\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle]$. In Sec. 2.2.2, we described a direct approach that consists of constructing a new polynomial $P_\mathbf{q}$ and returning Equation 2.11. However, as described in Sec. 2.2.2 and Sec. 2.3.1, 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. 2.2.2).

Lemma 2.3.4. *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

$$\frac{\alpha_{j_1} \cdots \alpha_{j_\ell} \partial^\ell P_\mathbf{q}}{\partial \alpha_{j_1} \cdots \partial \alpha_{j_\ell}} = \frac{\alpha_{j_1} \cdots \alpha_{j_\ell} \beta \partial^{\ell+1} P_\mathbf{q}}{\partial \alpha_{j_1} \cdots \partial \alpha_{j_\ell} \partial \beta} \tag{2.16}$$

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

$$\frac{\alpha_{j_1} \cdots \alpha_{j_\ell} \partial^\ell P_\mathbf{q}}{\partial \alpha_{j_1} \cdots \partial \alpha_{j_\ell}} = \frac{\beta \partial P_\mathbf{q}}{\partial \beta} \tag{2.17}$$

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 Equation 2.16

contain β and thus remain unaffected by applying the operator $\beta\partial/\partial\beta$.

(2) From item (1), we derive Equation 2.16; we prove now that the RHS of Equation 2.16 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 Equation 2.16 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 Equation 2.16. 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. □

Corollary 2.3.5. *(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,*

$$\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle] = \frac{n}{P} \frac{\alpha_{j_1} \cdots \alpha_{j_\ell} \partial^\ell P}{\partial\alpha_{j_1} \cdots \partial\alpha_{j_\ell}} \tag{2.18}$$

(2) Let $\mathbf{q}$ *be the query defined by a predicate as in Equation 2.15. Then,*

$$\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle] = \sum_{j_1 \in J_1 : \pi_{j_1} \Rightarrow \rho_1} \cdots \sum_{j_m \in J_m : \pi_{j_m} \Rightarrow \rho_m} \frac{n}{P} \frac{\alpha_{j_1} \cdots \alpha_{j_m} \partial^m P}{\partial\alpha_{j_1} \cdots \partial\alpha_{j_m}} \tag{2.19}$$

Proof. (1) Equation 2.18 follows from Equation 2.11, Equation 2.17, 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. 2.3.4 (1). □

In order to compute a query using Equation 2.19, 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 Equation 2.18 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 $\{\alpha_j : j \in J_i\}$ (Equation 2.8). Let ρ_i be any predicate on the attribute A_i. Then,

$$\sum_{j_i \in J_i : \pi_{j_i} \Rightarrow \rho_i} \frac{\alpha_{j_i} \partial P}{\partial \alpha_{j_i}} = P[\bigwedge_{j \in J_i : \pi_{j_i} \not\Rightarrow \rho_i} \alpha_j = 0] \tag{2.20}$$

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 Equation 2.19 and obtain the following, much simplified formula for answering a query $\mathbf{q}$, defined by a predicate of the form Equation 2.15:

$$\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle] = \frac{n}{P} P[\bigwedge_{i=1,m} \bigwedge_{j \in J_i : \pi_{j_i} \not\Rightarrow \rho_i} \alpha_j = 0] \tag{2.21}$$

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 2.3.6. *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 Figure 2.2. 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.*

2.4 System Optimizations

In Section 2.3, 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.

2.4.1 Building the Polynomial

Recall from Theorem 2.3.2 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 Figure 2.2 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 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 Figure 2.2), 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 Theorem 2.3.2. 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

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)^2 B̂_s^k)
    // add part iii to P
    for (group in satGrps) do
      P.addTerms(buildTerm(group))   } O(mN + B̂_s^k)
```

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.

Figure 2.3 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 Section 2.3, 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 $\rho_{iL} \wedge \rho_{iS} \not\equiv \texttt{false}$. 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 $\rho_{iL} \wedge \rho_{iS} \equiv \texttt{true}$. 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 Figure 2.2 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

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

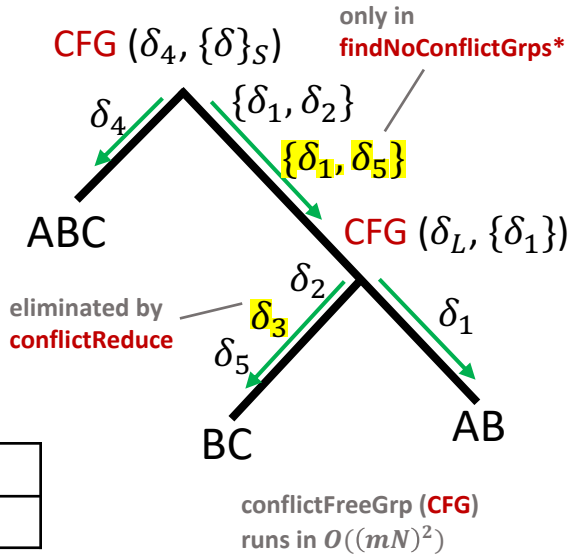


Figure 2.3: Figure of findNoConflictGrps and findNoConflictGrps* join algorithm for the three attribute sets AB, BC, and ABC. The highlighted statistics of δ_3 and $\{\delta_1, \delta_5\}$ are the difference between the unoptimized and optimized algorithms.

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 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 Figure 2.3. 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

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]))                 } O(mNB_a + B_a B̂_s)
  // add part iii to P                                    }
  for (group in multiDStats[idx]) do                      }
    P.addTerms(buildTerm(group))                          }
redMDStats = conflictReduce(multiDStats) — O(C(B_a,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]))               } O((2^{B_a} − B_a − 1) * (mN + B̂_s^{B_a}))
    // add part iii to P                                  }
    for (group in satGrps) do                             }
      P.addTerms(buildTerm(group[idx]))                   }
```

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 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(δ_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, Figure 2.4 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.

2.4.2 Polynomial Evaluation

Recall from Section 2.3.2 and Example 2.3.6 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

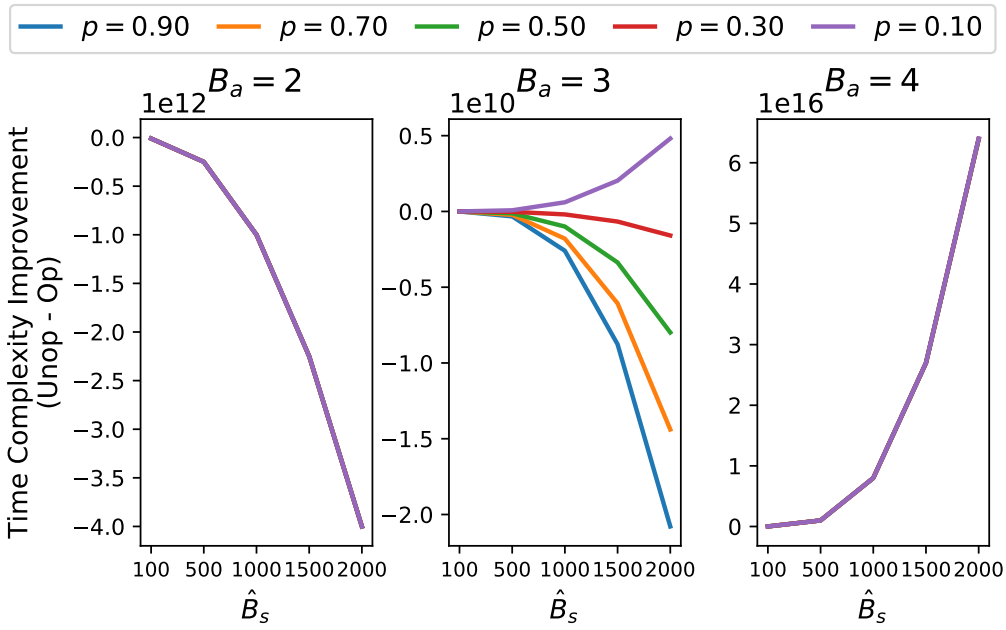


Figure 2.4: Algorithmic complexity improvement of the unoptimized algorithm over the optimized algorithm (complexity difference) for $mN = 5000$ and varying B_a, $\hat{B}_s$, and p. A positive number indicates the optimized algorithm is faster.

compressed polynomial can, at worst, have exponentially many variables (see Theorem 2.3.2).

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 2.3.6 which references Figure 2.2. 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 Section 2.4.1 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 Section 2.6.

2.5 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 reordering 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.

2.5.1 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 [122]. This means each multi-dimensional statistic predicate π_j is equivalent to a range predicate over two attributes $A_{i_1} \in [u_1, v_1] \wedge A_{i_2} \in [u_2, v_2]$. 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 Section 2.3.1, 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. 2.6. 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[6] 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 [39]. 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. 2.6.

Next, we assume for each attribute A_i, its domain D_i is ordered and viewed as an array such that $D_i[1] \leq D_i[2] \leq ....$ 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

[6] 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.

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 Theorem 2.3.2 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 Equation 2.22 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 $[l^x, m^x]$.

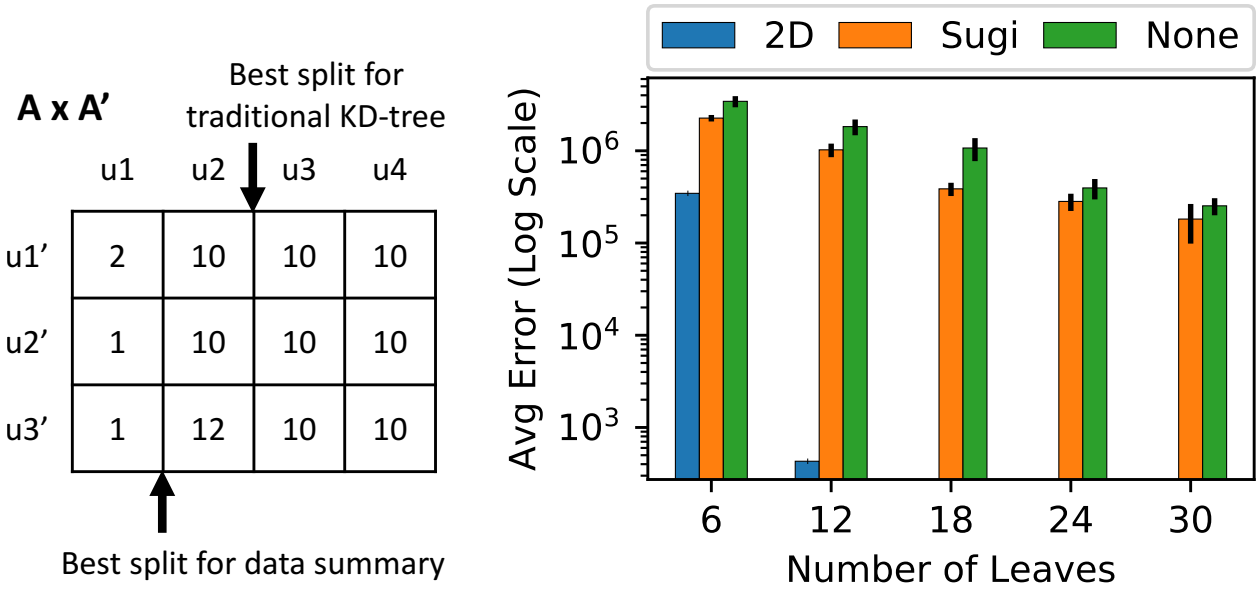


Figure 2.5: (a) Example K-D tree showing the traditional split on the median versus our split minimizing average error. (b) Comparison of not sorting, using **SUGI** sort, or using **2D** sort before running the K-D tree algorithm.

$$\arg\min_{m^x} \left[\sum_{(x,y)\in[l^x,m^x]\times[l^y,u^y]} (\mathcal{M}[x,y] - \bar{s_l})^2 + \sum_{(x,y)\in[m^x+1,u^x]\times[l^y,u^y]} (\mathcal{M}[x,y] - \bar{s_r})^2 \right]^{1/2}. \quad (2.22)$$

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 2.5a. 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 Equation 2.22 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 Section 2.6.5 we evaluate the three different heuristic selection techniques.

2.5.2 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 (Equation 2.22) 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 Figure 2.6. 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 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 Section 2.5.3). 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 $\text{KD}(\mathcal{M}) = \{[l_j^x, u_j^x] \times [l_j^y, u_j^y] : j = 1, B_s\}$. The K-D tree error is

$$\text{err}(\text{KD}(\mathcal{M})) \stackrel{\text{def}}{=} \frac{1}{B_s} \left[\sum_{(x,y) \in [l_j^x, u_j^x] \times [l_j^y, u_j^y]} (\mathcal{M}[x, y] - \bar{s}_j)^2 \right]^{1/2} \tag{2.23}$$

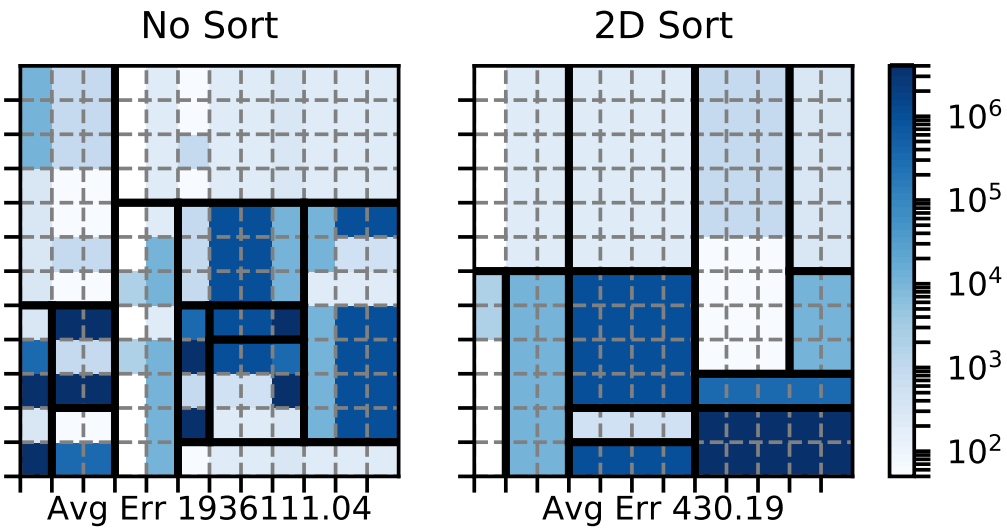


Figure 2.6: Plots showing the frequency heatmaps and the K-D trees built on data that is unsorted (left) and sorted using the **2D** sort algorithm (right). The average K-D tree leaf error is shown below.

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

$$\arg\min_{\pi_x,\pi_y} \text{err}(\text{KD}(\pi_x \mathcal{M} \pi_y)) \tag{2.24}$$

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

To solve this, we rely on heuristic techniques.

2.5.3 Heuristic Sorts

Inspired by the work in finding optimal matrix reorderings for data visualization and Rectangle Rule List minimization [91, 24, 17], we experiment with two different heuristic sort algorithms described in [91] to more optimally order $\mathcal{M}$ and reduce Equation 2.23. 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

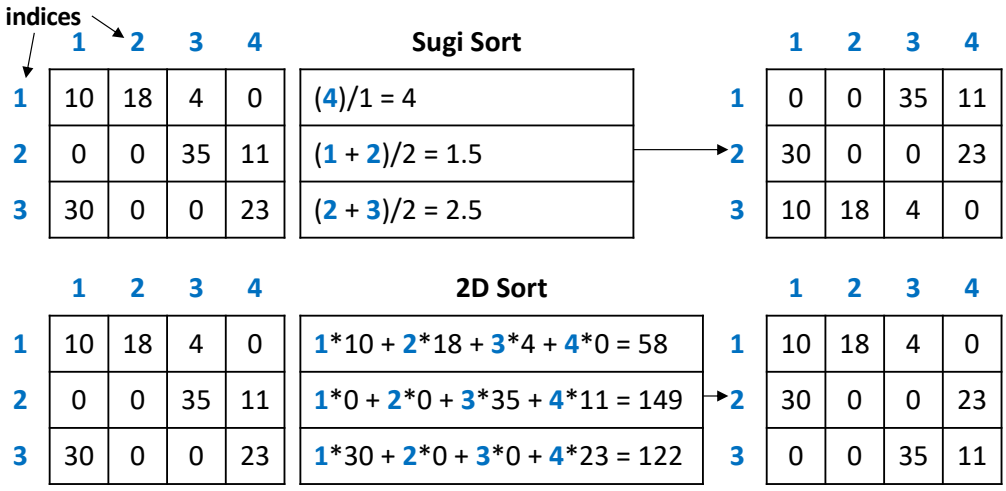


Figure 2.7: Sorting a matrix's rows by **2D** sort (top) and **SUGI** sort (bottom).

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, Figure 2.7 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 $\text{err}(\text{KD}(\mathcal{M})) = 0$ 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.

2.5b shows the difference in average K-D tree error (Equation 2.23) 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 Section 2.6.5 how using **2D** sort impacts the overall query error of our MaxEnt technique.

2.6 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.

2.6.1 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 Section 2.4.2) 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[7]. 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.

2.6.2 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. 2.2.3. 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 [4]. 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 Figure 2.8 shows the resulting active domain sizes.

The second dataset is 210 GB in size. It comprises N-body particle simulation data [76], which captures the state of astronomy simulation particles at different moments in time

[7]The maximum amount of memory used in experiments was approximately 40 GB, meaning a system this large is not required.

	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 2.8: 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.

(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 Figure 2.8 shows the resulting domain sizes.

2.6.3 Query Accuracy

We first compare EntropyDB using our best statistic selection techniques of **COMPOSITE** and **2D** sort (see Section 2.6.5) to uniform and stratified sampling on the flights dataset. We use one percent samples, which require 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,

	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 2.9: MaxEnt 2D statistics including in the summaries. The top row is the label of the MaxEnt method used in the graphs.

distance), 2C = (destination_state, distance), 3CF = (fl_time, distance)[8], 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 Section 2.5, 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. Figure 2.9 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:

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

```
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 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. 2.6.4 reports execution time for the larger data.

Figure 2.10 (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

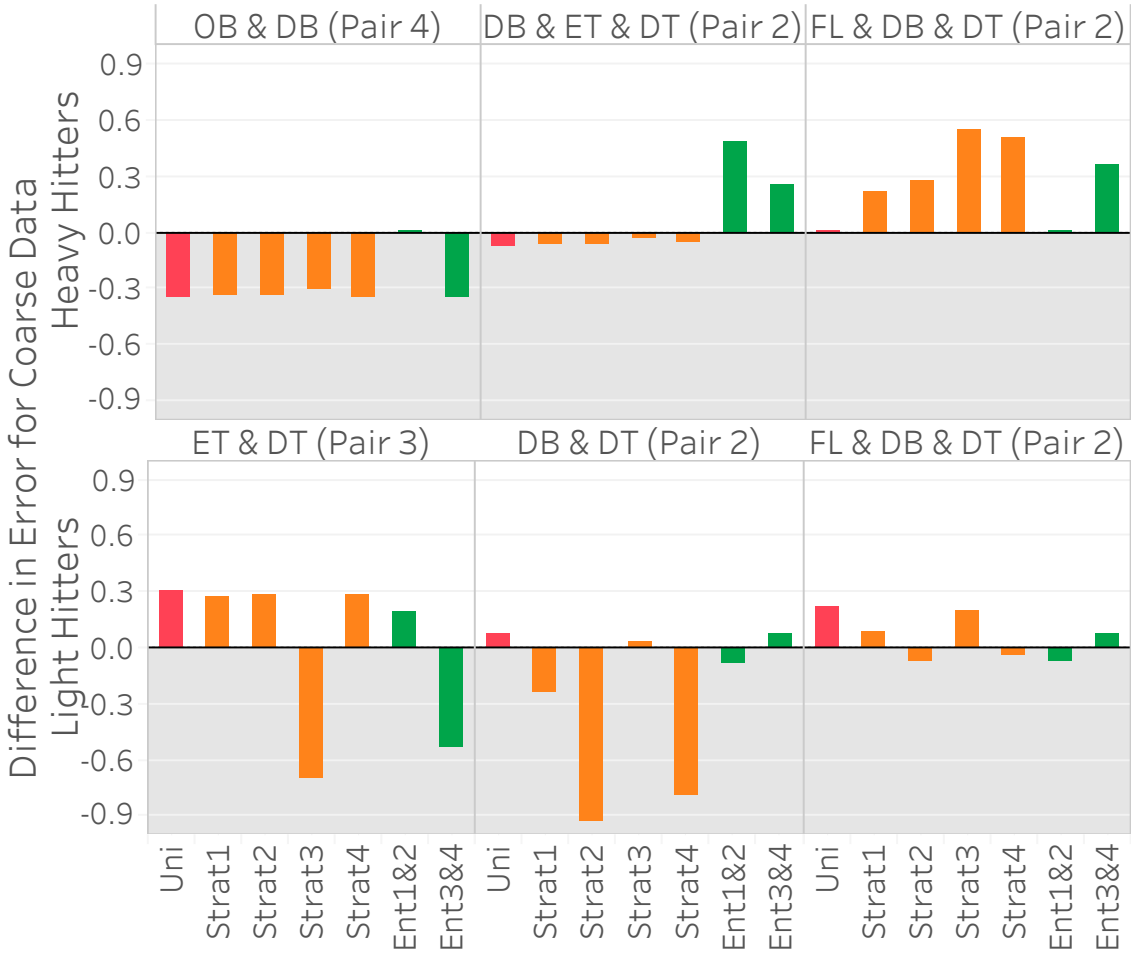


Figure 2.10: Query error difference between all methods and Ent1&2&3 over `FlightsCoarse`. The pair in parenthesis in the column header corresponds to the 2D statistic pair(s) used in the query template. For reference, pair 1 is (origin/OB, distance/DT), pair 2 is (dest/DB, distance/DT), pair 3 is (time/ET, distance/DT), and pair 4 is (origin/OB, dest/DB).

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.

Figure 2.10 (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

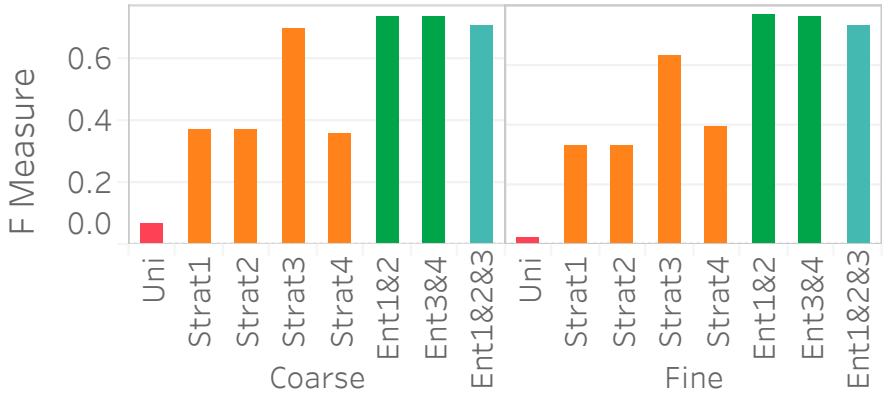


Figure 2.11: F measure for light hitters and null values over `FlightsCoarse` (left) and `FlightsFine` (right).

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 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, Figure 2.11 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.

2.6.4 Execution Times

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. 2.6.3, split into heavy hitters and light hitters. We record the query accuracy and execution time.

Figure 2.12 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 relatively 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

Figure 2.12: Query average error and execution time for three 4D selection queries on the `Particles` table. The stratified sample (orange) is stratified on (den, grp).

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.

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 Figure 2.13). When $B_a = 2$, we gather statistics over pair 1 and pair 2 (MaxEnt1&2),

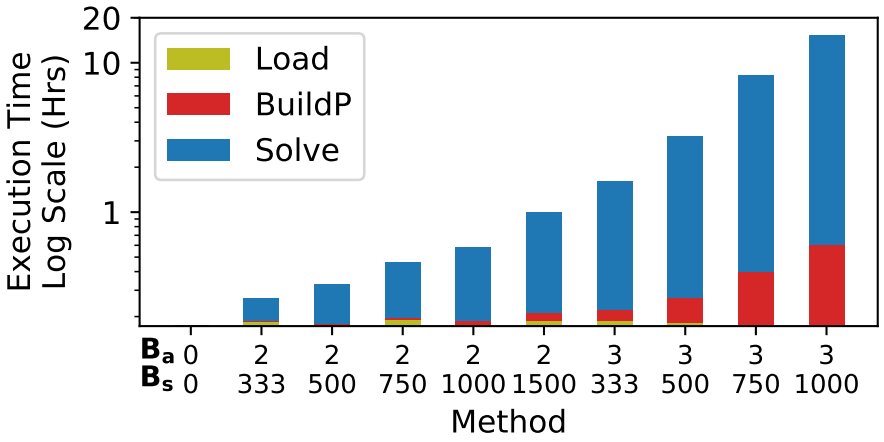


Figure 2.13: EntropyDB log scale execution times for loading the data, building the polynomial, and solving for the parameters for various configurations of B_a and B_s on `FlightsFine`.

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 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.

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.

Figure 2.14 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 Figure 2.6.4 (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.

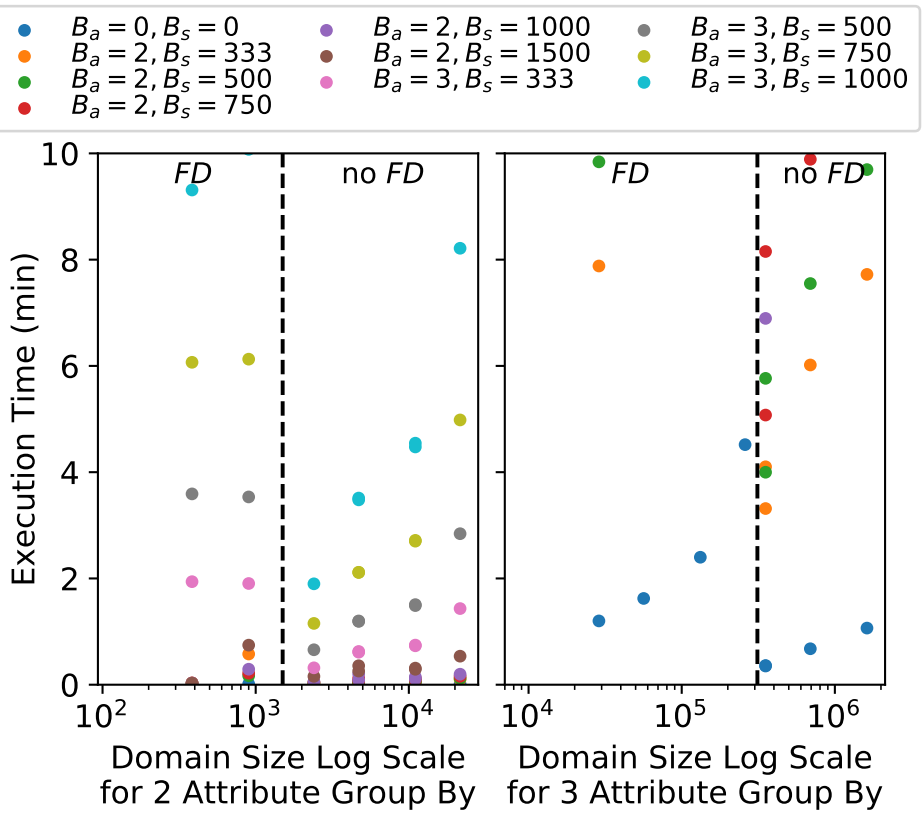


Figure 2.14: Log scale query execution times for 2- and 3-dimensional group-by queries versus size of query's active domain on `FlightsFine` for various configurations of B_a and B_s. The dashed line demarcates queries with `fl_date` as a group-by attribute and those that do not.

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 Figure 2.13. 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 Section 2.4.2. 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 Figure 2.2. 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.

2.6.5 Statistic Selection

Selection Technique

We evaluate the three different statistic selection heuristics, described in Section 2.5.1, 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 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.

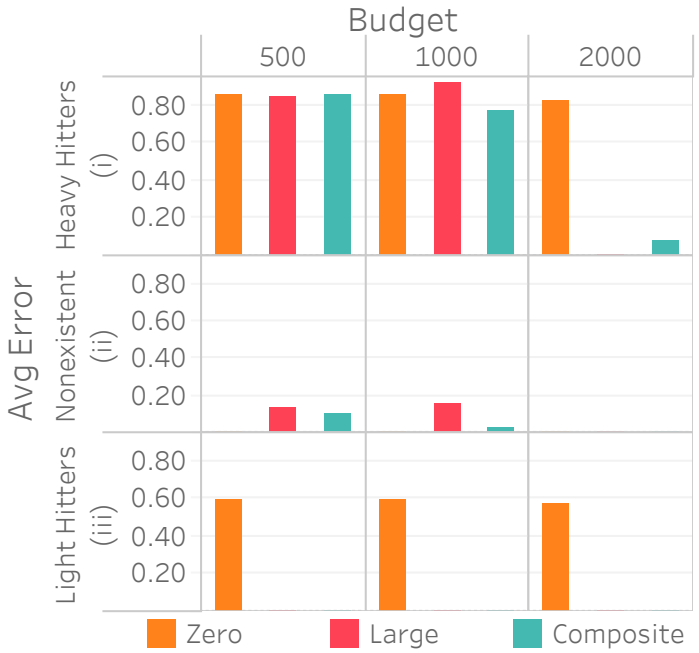


Figure 2.15: Illustration of query accuracy versus budget for the three different heuristics and three different selections: (i) selecting 100 heavy hitter values, (ii) selecting 200 nonexistent values, and (iii) selecting 100 light hitter values.

Figure 2.15 (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

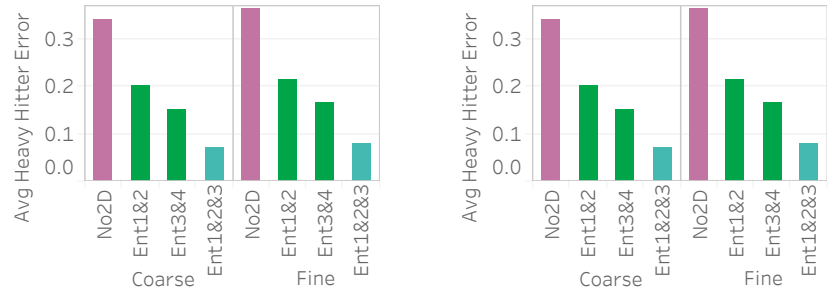
more of the light hitter values are nonexistent than it does with a smaller budget as less weight is distributed to the light hitter values.

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.

Statistic Accuracy

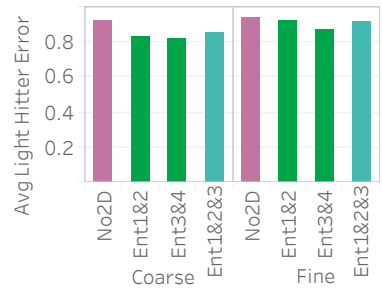
We now investigate, in more detail, how the different 2D statistic attribute choices and how presorting the matrix impacts query accuracy. We look at the query accuracy of the four different MaxEnt methods used in Figure 2.10 using both **2D** sort and no sort. We also include the MaxEnt method No2D 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 Section 2.6.3. 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.

Figure 2.16 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 attributes yields

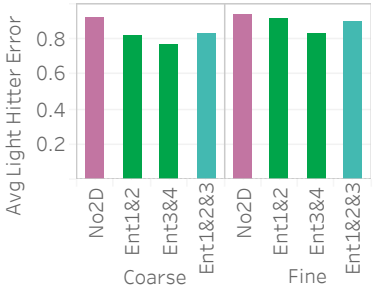


(a) Heavy Hitters No Sort

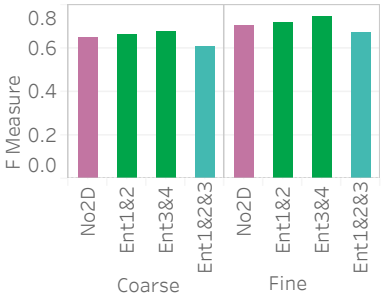
(b) Heavy Hitters **2D** Sort



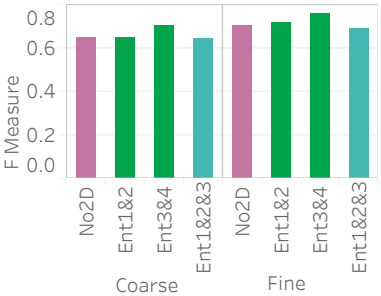
(c) Light Hitters No Sort



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



(e) F Measure No Sort



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

Figure 2.16: (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).

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 Figure 2.17. 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.

2.7 Discussion

2.7.1 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

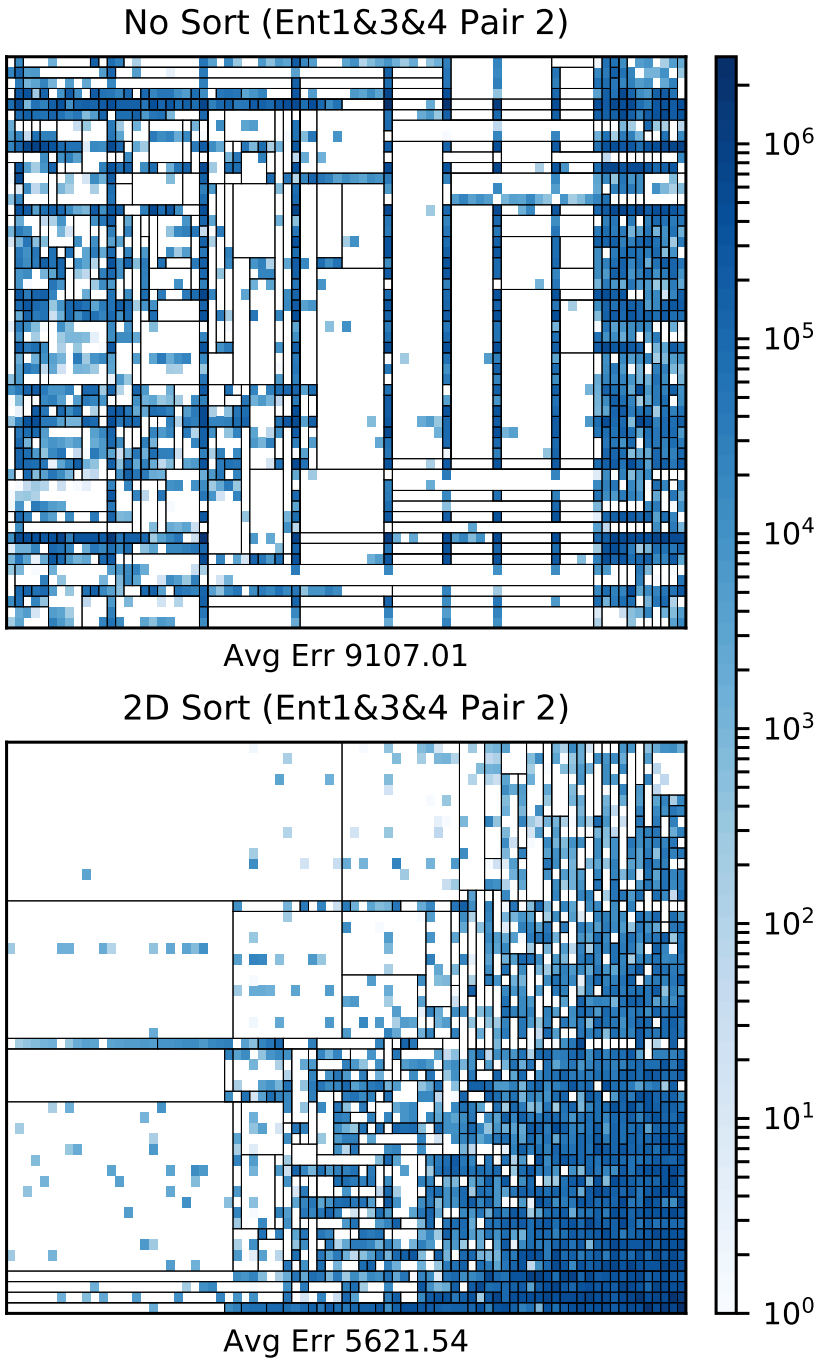


Figure 2.17: Plots showing the frequency heatmap of the pair 2 attributes of MaxEnt1&3&4 of `FlightsCoarse` that is unsorted (left) and sorted using the **2D** sort algorithm (right). The average K-D tree error is shown below.

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 [48]. 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 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.

2.7.2 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.

Joins

In Section 2.2, 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 Section 2.2.

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.

$$\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle] = \sum_{d_1 \in D_{j_{1,2}}} \cdots \sum_{d_{r-1} \in D_{j_{r-1,r}}} \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_1 \rangle] \ldots \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_r \rangle]$$
$$\text{s.t. } \pi_{\mathbf{q}'} = \pi_{\mathbf{q}} \wedge (R_1.A_{j_{1,2}} = d_1) \wedge (R_2.A_{j_{1,2}} = d_1) \wedge \ldots \wedge (R_r.A_{j_{r-1,r}} = d_{r-1}).$$

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 $\rho \equiv \mathit{true}$ 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 Figure 2.14, 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 Section 2.5 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 2.7.1). 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 2.7.1. *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 Equation 2.1.

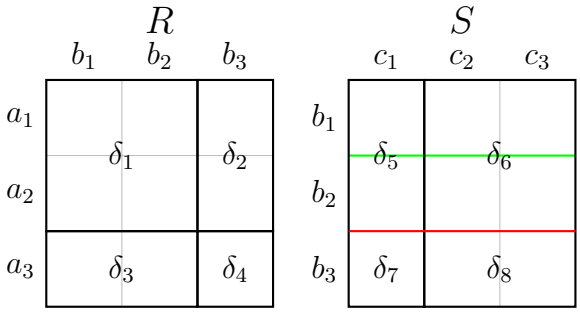
By our naïve strategy, the answer to $\mathbf{q}$ is

$$\mathbb{E}[\langle \mathbf{q}, \mathbf{I_R} \bowtie \mathbf{I_S} \rangle] = \sum_{b \in [b_1, b_3]} \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_R \rangle] \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_S \rangle]$$
$$s.t.\ \pi_{\mathbf{q}'} = (A = a_1 \wedge C = c_1 \wedge B = b).$$

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

$$\begin{aligned}
\mathbb{E}[\langle \mathbf{q}, \mathbf{I} \rangle] &= \sum_{d_1 \in D_{j_{1,2}}} \cdots \sum_{d_{r-1} \in D_{j_{r-1,r}}} \prod_{i=1}^{r} \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_i \rangle] \\
&= \sum_{d_1 \in D_{j_{1,2}}} \cdots \sum_{g_k^{r-1,r}} \sum_{d_{r-1} \in g_k^{r-1,r}} \prod_{i=1}^{r} \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_i \rangle] \\
&= \sum_{d_1 \in D_{j_{1,2}}} \cdots \sum_{g_k^{r-1,r}} \mathbb{E}[\langle \mathbf{q}'_r, \mathbf{I}_r \rangle] \sum_{d_{r-1} \in g_k^{r-1,r}} \prod_{i=1}^{r-1} \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_i \rangle] \\
&= \sum_{d_1 \in D_{j_{1,2}}} \cdots \sum_{g_k^{r-1,r}} \left[|g_k^{r-1,r}| * \mathbb{E}[\langle \mathbf{q}'_{r-1}, \mathbf{I}_{r-1} \rangle] * \mathbb{E}[\langle \mathbf{q}'_r, \mathbf{I}_r \rangle] * \prod_{i=1}^{r-2} \mathbb{E}[\langle \mathbf{q}', \mathbf{I}_i \rangle] \right]
\end{aligned}$$

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

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 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.

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})
```

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.

Updates

Another key assumption made in Section 2.2 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.

2.7.3 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 [116, 45]?

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 Equation 2.11, 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 2.7.2. *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 Section 2.2.2 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 Equation 2.10, we get

$$\begin{aligned}
(P_{\mathbf{t}_\ell})^n &= \left(\sum_{i=1,d} \prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{t}_i \rangle} \beta^{\langle \mathbf{t}_\ell, \mathbf{t}_i \rangle} \right)^n \\
&= \left(\prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{t}_\ell \rangle} \beta + \sum_{\substack{i=1,d \\ i \neq \ell}} \prod_{j=1,k} \alpha_j^{\langle \mathbf{c}_j, \mathbf{t}_i \rangle} \beta^{\langle \mathbf{t}_\ell, \mathbf{t}_i \rangle} \right)^n \\
&= \left(\left(\beta \frac{\partial P_{\mathbf{t}_\ell}}{\partial \beta} \right) + \left(P_{\mathbf{t}_\ell} - \beta \frac{\partial P_{\mathbf{t}_\ell}}{\partial \beta} \right) \right)^n . \qquad (2.25)
\end{aligned}$$

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 Equation 2.6. From Equation 2.11 and since $\beta = 1$ and $n = 1$, we get

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

□

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.

2.7.4 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 [100, 120, 124, 77]. This can be

seen by transforming Equation 2.2 into exponential form

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

Further, this exponential form is equivalent to the probability distribution defined over an exponential family Markov Network or, more generally, factor graph [131]. Markov networks are factor graphs where the factors representing the parameterization of the probability distribution are defined solely on maximal 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 Section 2.2.3 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 Equation 2.5). This simplification allows for drastic performance benefits in terms of solving and query answering (Section 2.3).

2.8 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.

Chapter 3

THEMIS: SAMPLE DEBIASING IN AN OPEN WORLD DATABASE SYSTEM

The research question addressed in this chapter is how to develop a system that automatically debiases a sample of data using population aggregates for population query answering.

As mentioned in Chapter 1, data samples are increasingly easy to access and analyze with the help of websites and data repositories. Additionally, data analytic toolkits, like Python, are becoming more mainstream. These two factors have lead to data science becoming tightly coupled with sample analysis.

Modern data scientists, however, face the added challenge that the data samples they seek to analyze are not always an accurate representation of the population they are sampled from. For example, social scientists today study migration patterns from Twitter samples [133], but Twitter users are not a uniform random sample of all people. This phenomenon is known as sample selection bias [42] and is problematic because it can lead to inaccurate analyses.

Correcting this bias, however, is difficult because the sampling mechanism in today's data sources, i.e., the probability of some population tuple being included in the sample, is typically not known. This means common techniques such as the Horvitz-Thompson estimator [22] (see Section 3.3.1) are not applicable.

Population aggregates can facilitate data debiasing by providing information on what the true distribution of the population is, but the debiasing process remains tedious and error prone. There is no general, automatic technique or system for debiasing using aggregates. With the ultimate goal of answering queries approximately over the population, data scientists are forced to manually implement one-off, specialized solutions [134] tailored towards specific datasets, such as census reports [97].

In this chapter, we present THEMIS, which is, to our knowledge, the first open world database management system (DMBS) that automates and encapsulates the debiasing process. The data scientist simply inserts a sample and aggregates and then asks queries, getting approximate results as if the queries were issued on the population. THEMIS is an open world DBMS as it inherently treats relations as samples and assumes tuples not in the sample could still exist.

To achieve our goal, at the heart of our system, we develop and combine two different debiasing techniques: reweighting the sample and learning the probability distribution of the population. The former allows us to more accurately answer heavy hitter queries while the later ensures we can answer queries about tuples that may not exist in the sample.

For sample reweighting, we investigate two different approaches: modifying linear regression and applying an existing aggregate fitting procedure. For learning the probability distribution, we utilize Bayesian networks to build an approximate population probability distribution. The novelty of our system is in not only building two separate debiasing techniques but also combining them into one unified system for query answering. Depending on the query issued, we automatically choose which approach is best using a simple, effective heuristic.

We build a prototype database system called THEMIS, named after the Greek titan for balance and order who is often seen holding a set of scales and accurately represents our goal of rebalancing data. THEMIS treats relations as samples and automatically corrects for sample selection bias using population-level aggregates. We evaluate THEMIS on three datasets to show that THEMIS is more accurate at answering point queries and top-k queries than standard uniform reweighting, linear regression reweighting (Figure 3.17), and a variety of Bayesian network probabilistic approaches (Figure 3.16).

In summary, the contributions of this chapter are as follows:

- The first open world database system that takes a sample and population aggregates and automatically debiases the data for approximate population query answering (Sec-

tion 3.2).

- The development and application of debiasing techniques and a novel hybrid approach integrating them (Section 3.3).
- Two optimization techniques for faster preprocessing time: population aggregate pruning and model simplification (Section 3.4).
- Detailed experiments on three datasets showing that THEMIS achieves a 70 percent improvement in the median error when compared to a naïve scaling approach when asking about heavy hitter tuples (Table 3.7, Section 3.5). We further show THEMIS is robust to differences in the support of the sample and the population.

The chapter is organized as follows. Section 3.1 gives a motivating example for THEMIS, Section 3.2 gives the high level model of THEMIS, Section 3.3 describes our technique in detail, and Section 3.4 discusses our optimization. Finally, Section 3.5 provides experimental results.

3.1 Motivating Example

A data scientist is trying to estimate the number of flights under 30 min in different states of the United States in a year. She has a sample of all flights in the United States skewed towards four major states, but she does not know how badly it is skewed. Further, she has access to how many flights in total leave from each state. She decides to analyze this data and focus only on short flights on either the East or West Coast of the country.

Being a database user, she ingests the data into a SQL database and prepares to analyze it. As this dataset is a sample, she has three choices for how to prepare her data for analysis: do nothing, ignore skew and uniformly rebalance, or use state information to reweight flights based on the number of flights leaving each state. For the second option, she knows there are 7 million flights in the United States per year but only 700,000 in her sample. Therefore, she adds a `weight` attribute to the dataset and gives each tuple a weight of 10, indicating the each tuple in her sample represents 10 tuples in the real world. For the third option, if

Query	True	Raw	Unif	US State	THEMIS
CA	7855	2846	28460	7843	7843
FL	2	1	10	3	3
OH	119	1	10	70	70
ME	2	0	0	0	3

Table 3.1: Query results of the data scientists using the raw sample, a uniformly scaled sample, a state-scaled sample, and THEMIS.

she knows there are N flights leaving from some state per year but only n leaving that state in her sample, she sets the weight of each flight from that state to be N/n.

Having done preprocessing work, she is ready to ask for the number of quick flights from various states. She starts issuing point queries of the form

```
SELECT SUM(weight) AS num_flights
FROM flights WHERE flight_time <= 30 min
AND origin_state = '<state>';
```

The results of a few state queries are shown in Table 3.1 where Raw represents option one, Unif represents option two, US State represents option three, and THEMIS represents our system's answer. THEMIS and US State use the single aggregate to produces more accurate answers than Raw and Unif because they are correcting for the fact that some flights leaving the four major states are overrepresented in the samples. More importantly, THEMIS does the rebalancing automatically, which will become time consuming to do manually for more complex aggregates. THEMIS is also able to answer queries about tuples not in the sample, like ME.

3.2 THEMIS *Model*

We now describe our data debiasing setup and give an overview of THEMIS (see Figure 3.1). At a high level, THEMIS uses a sample and population aggregate data to build a model which approximately answers population queries. We use the term *model* because it encapsulates that we use both a reweighted sample and a probabilistic model to answer queries. Both

Symbol	Definition
P	population
S	sample
n	population size
m	number attributes
n_S	sample size
N_i	size of domain of A_i
B	$\|\Gamma\|$, aggregate budget
d_i	dimensionality of aggregate i
M_i	$\|\Gamma_i\|$
Γ	$\{G_{\boldsymbol{\gamma}_i,\text{COUNT}(*)}(P)\}$
Γ_i	$\{(\boldsymbol{a}_{i,k}, c_{i,k}) : k = 1, \ldots, M_i\}$
$\overline{\Gamma}_i^A$	$[\boldsymbol{a}_{i,k}], k = 1, \ldots, M_i$ matrix
$\overline{\Gamma}_i^C$	$[c_{i,k}], k = 1, \ldots, M_i$ vector

Table 3.2: THEMIS common notation.

techniques treat the aggregates as constraints to be satisfied.

Note that the population aggregates do not need to be exact. They may contain errors, be computed at different times, or be purposely perturbed. For example, the 2020 US census will add random noise to their reports to be differentially private [52]. THEMIS will still treat these aggregates as marginal constraints to be satisfied.

We assume there is a well defined, but unavailable population P of (approximate) size n with m attributes $\mathcal{A} = \{A_1, \ldots, A_m\}$ (attributes listed in Table 3.2. P is unavailable because either it does not exist (e.g., a dataset of all graduate students in the US) or is not released to the public (e.g., a hospital's private medical data). The active domain of each attribute A_i, of size N_i, is assumed to be discrete and ordered[1].

We assume there is a sample S drawn independently but not uniformly from P of size n_S such that for each tuple $t \in P$, t has probability $\Pr_S(t)$ of being included in S. The subscript S indicates the sampling probability (also called sampling mechanism or propensity score). This probability, however, is not known a priori.

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

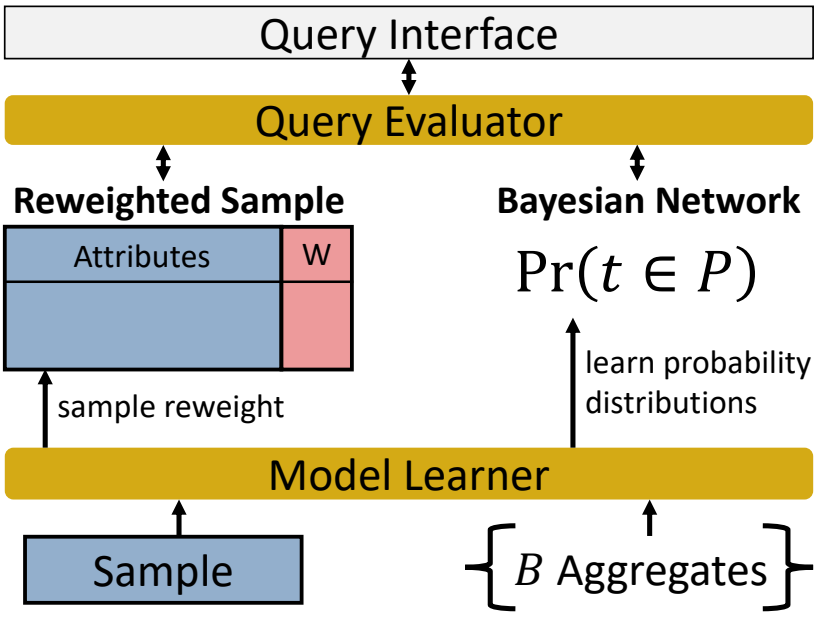


Figure 3.1: THEMIS architecture.

Lastly, we have Γ, a set of results of B aggregate `COUNT(*)` queries of various dimensions computed over the population denoted

$$\Gamma = \{G_{\boldsymbol{\gamma}_i,\text{COUNT}(*)}(P) : i = 1, B\}$$

where $G_{\boldsymbol{\gamma}_i,\text{COUNT}(*)}(P)$ is an aggregate query of dimension d_i; i.e., $\boldsymbol{\gamma}_i \subseteq \mathcal{A}$ (see Example 3.2.1). Each aggregate query Γ_i returns a set of M_i attribute value-count pairs denoted

$$\Gamma_i = \{(\boldsymbol{a}_{i,k}, c_{i,k}) : k = 1, \ldots, M_i\}$$

where $\boldsymbol{a}_{i,k}$ is the vector of d_i attribute values associated with group k of aggregate i, and $c_{i,k}$ is the group's count.

If the aggregates are exact, for all i, $\sum_k c_{i,k} = n$. Otherwise, $\sum_k c_{i,k}$ approximates n. Further, N_j, the number of distinct values of attribute A_j, can be calculated from any Γ_i if $A_j \in \boldsymbol{\gamma}_i$ or, if there is no such i, S.

We do not require the aggregates to cover the attributes, meaning $\bigcup_{i=1,B} \boldsymbol{\gamma}_i \subseteq \mathcal{A}$; i.e., we do not assume that $\bigcup_{i=1,B} \boldsymbol{\gamma}_i = \mathcal{A}$.

When we wish to refer to all the group values and counts of aggregate i separately, we will use $\overline{\Gamma}_i^A$ and $\overline{\Gamma}_i^C$, respectively.

Example 3.2.1. *Following the example from Section 3.1, assume the population P and sample S are the following sets of domestic flights in the United States.* `date` *is the month and* `o_st` *and* `d_st` *are origin and destination states, respectively.*

$P =$

`date`	`o_st`	`d_st`
01	*FL*	*FL*
01	*FL*	*FL*
02	*FL*	*NY*
01	*NC*	*FL*
02	*NC*	*NY*
02	*NC*	*NY*
02	*NC*	*NY*
01	*NY*	*FL*
01	*NY*	*NC*
02	*NY*	*NY*

$S =$

`date`	`o_st`	`d_st`
01	*FL*	*FL*
01	*FL*	*FL*
02	*NC*	*NY*
01	*NY*	*NC*

Let $\Gamma = \{\Gamma_1, \Gamma_2\}$ with $d_1 = 1$ and $d_2 = 2$ be the following two aggregate queries.

$$\Gamma_1 = G_{\texttt{date},\texttt{COUNT(*)}}(P) = \{([01], 5), ([02], 5)\}$$

$$\Gamma_2 = G_{\texttt{o\_st},\texttt{d\_st},\texttt{COUNT(*)}}(P) =$$

$$\{([FL, FL], 2), ([FL, NY], 1), ([NC, FL], 1),$$

$$([NC, NY], 3), ([NY, FL], 1), ([NY, NC], 1), ([NY, NY], 1)\}.$$

In this case, $n = 10$*,* $B = 2$*, and the* Γ*s are set as*

$$\overline{\Gamma}_1^A = \begin{bmatrix} 01 \\ 02 \end{bmatrix} \qquad \overline{\Gamma}_1^C = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

$$\overline{\Gamma}_2^A = \begin{bmatrix} FL & FL \\ FL & NY \\ NC & FL \\ NC & NC \\ NY & FL \\ NY & NC \\ NY & NY \end{bmatrix} \qquad \overline{\Gamma}_2^C = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 3 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

The ordering of the rows $\overline{\Gamma}_i^A$ *and* $\overline{\Gamma}_i^C$ *does not matter, as long as it is consistent.*

We are given a user query Q over the population. While Q can be any SQL query, to study the improvement in accuracy by using THEMIS, we only focus on two types of queries: simple point and top-k queries.

A point query is of the form

SELECT COUNT(⋆)
FROM P
WHERE $A_{i_1} = v_{i_1}$ **AND** ... **AND** $A_{i_\ell} = v_{i_\ell}$**;**

where the query is asking for the tuple of tuples satisfying selection conditions of ℓ attributes.

A top-k query is of the form

SELECT A_{i_1}**,** ...**,** A_{i_ℓ}
FROM P
GROUP BY A_{i_1}**,** ...**,** A_{i_ℓ}
ORDER BY COUNT(⋆) **DESC;**

where the query is asking for the k largest ℓ attribute values in terms of count.

As we do not have P, we need to estimate $Q(P)$ given S and Γ. To do so, we build a model $\mathcal{M}(\Gamma, S)$ such that $Q(\mathcal{M}(\Gamma, S))$ is an approximate answer to $Q(P)$.

3.3 Data Debiasing

We are now ready to present how THEMIS builds $\mathcal{M}(\Gamma, S)$. THEMIS has two components: a reweighted sample and a probabilistic model. We present each technique and then describe how THEMIS merges them into a unique hybrid approach to answer $Q(P)$ approximately.

3.3.1 Sample Reweighting

In sample reweighting, each tuple $t \in S$ gets assigned a weight $w(t)$ representing the number of tuples t represents in P. For example, if S is a ten percent uniform random sample, each tuple in S will get $w(t) = 10$ because each tuple represents ten tuples from P.

Queries on P get transformed to run on weighted tuples by, for example, translating `COUNT(*)` to be `SUM(weight)`. If the sampling mechanism, $\Pr_S(t)$, is known, we can use the Horvitz-Thompson estimator which reweights each tuple by $w(t) = 1/\Pr_S(t)$ [42, 96].

The challenge is that we do not have the sampling mechanism. A simple approach is to perform uniform reweighting by setting $w(t)$ to be $|P|/|S|$. As we show in Section 3.5, when the sample is biased, this achieves low accuracy. To correct for the bias, we present two solutions for learning $w(t)$ using the sample S and aggregate information Γ. The first technique is to adapt linear regression, and the second is to apply Iterative Proportional Fitting (IPF) [73, 89].

Linear Regression Reweighting

Our linear regression reweighting technique is inspired by propensity score analysis, a technique commonly used to evaluate the effect of a medical treatment on an observational (nonrandomized) study [112, 95, 18]. The propensity score is the probability of a patient being assigned to a particular treatment group given patient characteristics (attributes).

This score can be used to rebalance the data to make the treatment assignment become independent of the patient characteristics.

We make the same assumption as propensity score analysis that a tuple's weight linearly depends on the attributes of t; i.e., the weight of a tuple is a linear combination of its attributes. In other words, let $\boldsymbol{t}^{0/1}$ represents the one-hot encoded tuple t where each categorical variable A_i is replaced by N_i binary indicator variables for each value in the active domain (see Example 3.3.1). Then $w(t) = \boldsymbol{t}^{0/1} \cdot \boldsymbol{\beta}$ where $\boldsymbol{\beta}$ is a vector of weights. Note that for the rest of this section, we use m to refer to the number of attributes covered by the aggregates and only use those attributes for learning the weight.

Example 3.3.1. *Continue with Example 3.2.1. The one-hot encoded version of the first tuple in S, $(01, FL, FL)$, is*

$$\begin{array}{cccccccc} d_{01} & d_{02} & o_{FL} & o_{NC} & o_{NY} & d_{FL} & d_{NC} & d_{NY} \\ \left[\begin{array}{cccccccc} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{array}\right]. \end{array}$$

To solve for $\boldsymbol{\beta}$, S gets represented by a $n_S \times m^{0/1}$ matrix, $\boldsymbol{X}_S$, where $m^{0/1} = \sum_{i=1}^{m} N_i + 1$ (the plus one is because we use the standard formulation of adding a column of ones to represent the intercept). $\boldsymbol{y}$ is $\overline{\Gamma}_1^C \oplus \ldots \oplus \overline{\Gamma}_B^C$ where $\oplus$ represents row-wise concatenation (equivalent to vertically stacking the vectors). In this case, $\boldsymbol{y}$ is a column vector of size $\sum_{i=1}^{B} M_i$ containing all the aggregate queries' count values; i.e.

$$\boldsymbol{y} = \begin{bmatrix} c_{1,1} & \cdots & c_{1,M_1} & \cdots & c_{B,1} & \cdots & c_{B,M_B} \end{bmatrix}^T.$$

Let $\boldsymbol{X}$ be the matrix product of $\boldsymbol{G}^{0/1}\boldsymbol{X}_S$ where $\boldsymbol{G}^{0/1}$ is a 0/1 matrix with $\sum_{i=1}^{B} M_i$ rows and n_S columns (see Example 3.3.2). $\boldsymbol{G}^{0/1}$ is an incidence matrix where row r and column c is 1 if row c of $\boldsymbol{X}_S$ participates in the rth group by result; i.e., if the rth attribute value from

$\overline{\Gamma}_1^A \oplus \ldots \oplus \overline{\Gamma}_B^A$ is in row c of S. We then solve

$$[\boldsymbol{G}^{0/1}\boldsymbol{X}_S]\boldsymbol{\beta} = \boldsymbol{y}. \tag{3.1}$$

In the case an entire row of $\boldsymbol{G}^{0/1}\boldsymbol{X}_S$ is all zeros, which happens with missing values in S, we drop that row and its associated value in $\boldsymbol{y}$.

Departing from standard linear regression solving techniques, we solve Equation 3.1 using a constrained least squares formulation to constrain $\boldsymbol{\beta}$ to be strictly positive. This enforces that each tuple in the sample gets $w(t) \geq 0$ and is represented in the population.

Further, as we want to avoid $w(t) = 0$, we add an additional row of $[n_S, 0, \ldots, 0]$ with $\sum_{i=1}^{m} N_i$ zeros to the matrix $\boldsymbol{G}^{0/1}\boldsymbol{X}_S$ and add the associated value of n_S to $\boldsymbol{y}$. This encourages the intercept value to be positive, which will force every tuple to get some positive weight (since the $\boldsymbol{\beta}$ parameters are already positive). Note, as we just want to influence the intercept value, we cannot achieve this by adding a row of ones to $\boldsymbol{G}^{0/1}$ because this will result in an additional row of n_S followed $\sum_{i=1}^{m} N_i$ non-zero values added to $\boldsymbol{G}^{0/1}\boldsymbol{X}_S$.

Lastly, if a $w(t)$ does get set to 0, we set $w(t) = 1$.

Example 3.3.2. *Continuing with Example 3.2.1. The one-hot encoded version of S, $\boldsymbol{X}_S$, is*

$$\begin{array}{c} \begin{array}{cccccccccc} 1s & d_{01} & d_{02} & o_{FL} & o_{NC} & o_{NY} & d_{FL} & d_{NC} & d_{NY} \end{array} \\ \left[\begin{array}{cccccccccc} 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{array}\right]. \end{array}$$

Our aggregate matrix is

$$\boldsymbol{G}^{0/1} = \left[\begin{array}{cccc|l} 1 & 1 & 0 & 1 & \texttt{date} = 01 \\ 0 & 0 & 1 & 0 & \texttt{date} = 02 \\ 1 & 1 & 0 & 0 & \texttt{o\_st} = FL \;\&\; \texttt{d\_st} = FL \\ 0 & 0 & 0 & 0 & \texttt{o\_st} = FL \;\&\; \texttt{d\_st} = NY \\ 0 & 0 & 0 & 0 & \texttt{o\_st} = NC \;\&\; \texttt{d\_st} = FL \\ 0 & 0 & 1 & 0 & \texttt{o\_st} = NC \;\&\; \texttt{d\_st} = NY \\ 0 & 0 & 0 & 0 & \texttt{o\_st} = NY \;\&\; \texttt{d\_st} = FL \\ 0 & 0 & 0 & 1 & \texttt{o\_st} = NY \;\&\; \texttt{d\_st} = NC \\ 0 & 0 & 0 & 0 & \texttt{o\_st} = NY \;\&\; \texttt{d\_st} = NY \end{array}\right]$$

where the right-most column shows the group attributes corresponding to the row in the matrix. After $\boldsymbol{G}^{0/1}\boldsymbol{X}_S$ *is calculated, we add the row of* $[4, 0, \ldots, 0]$ *at the bottom. Finally, our solution vector is*

$$\boldsymbol{y} = \begin{bmatrix} 5 & 5 & 2 & 1 & 1 & 3 & 1 & 1 & 1 & 4 \end{bmatrix}^T$$

where the final 4 is from adding the n_S *constraint to* $\boldsymbol{y}$*.*

With these two changes, we solve for $\boldsymbol{\beta}$ and $w(t) = \boldsymbol{t}^{0/1} \cdot \boldsymbol{\beta}$. The final processing step is to modify $w(t)$ so that $\sum_{t \in S} w(t) = n$. This is a simple multiplicative update to each $w(t)$ so that $w(t) = \frac{n}{\sum_{t \in S} w(t)} w(t)$. We do this to sum-normalize the weights to correctly reflect to true size of the population *after* learning $w(t)$. Note that uniform reweighting is equivalent to setting $w(t) \equiv 1$ before sum-normalizing.

Iterative Proportional Fitting

An alternative approach to finding $w(t)$ is to assume every $w(t)$ is independent and can be solved for directly; i.e., $w(t)$ is not a function of its attributes. Inspired by the technique of population synthesis in demography, we apply a technique called Iterative Proportional

Fitting (IPF) [89, 23, 97, 117, 54, 47] to solve for $w(t)$. While IPF is not new, our approach of using IPF for arbitrary data debiasing is novel.

To briefly review IPF, IPF is a simple iterative procedure for calibrating sample weights to match given population aggregates and is traditionally used to reweight representative microsamples of some population to aggregate census reports. For each individual aggregate, if that aggregate is not satisfied in the sample, the weights of the participating tuples are rescaled to satisfy the selected aggregate. The procedure continues to iterative over aggregates until all aggregates are satisfied. It converges to a satisfactory scaling if such a scaling exists[2]. If no scaling exists, the algorithm may not converge and can only give an approximate reweighting.

The iterative algorithm begins by building the same incidence matrix, $\boldsymbol{G}^{0/1}$, as before, where each row represents a single constraint and each column represents a tuple in S. $\boldsymbol{y}$ is the vector of all aggregate queries' count values. With IPF, however, we have no $\boldsymbol{X}_S$. Instead, we have a n_S sized vector $\boldsymbol{w}$ of the weights of each tuple; i.e., $\boldsymbol{G}^{0/1}\boldsymbol{w} = \boldsymbol{y}$. At each iteration, a value in $\boldsymbol{w}$ is updated so that its associated aggregate constraint is satisfied (see Example 3.3.3).

The pseudocode for IPF is shown in Algorithm 5 where $[j]$ represents getting row j for matrices and element j for vectors. At each iteration, if the dot product of the jth row of $\boldsymbol{G}^{0/1}$ with $\boldsymbol{w}$ does not equal the jth element in $\boldsymbol{y}$, the weights are scaled so that the constraint is satisfied. Note that only the weights participating in the aggregate, i.e., with nonzero $\boldsymbol{G}^{0/1}[j]$ values, are updated.

Example 3.3.3 gives an example of the IPF algorithm and further demonstrates how missing values in S can prevent IPF from converging.

[2]IFP is the same algorithm as in matrix scaling, the RAS algorithm, and biproportional fitting [114, 89].

Algorithm 5 IPF [97]

$iter \leftarrow 1$
while not converged or $iter < maxIter$ **do**
  for $j = 1$ to $\sum_{i=1}^{B} M_i$ **do**
    if $\boldsymbol{G}^{0/1}[j] \cdot \boldsymbol{w} \neq \boldsymbol{y}[j]$ **then**
      $s \leftarrow \frac{\boldsymbol{y}[j]}{\boldsymbol{G}^{0/1}[j] \cdot \boldsymbol{w}}$
      for $i = 1$ to n_S **do**
        if $\boldsymbol{G}^{0/1}[j][i] = 1$ **then**
          $\boldsymbol{w}[i] \leftarrow s * \boldsymbol{w}[i]$
        end if
      end for
    end if
  end for
  $iter \leftarrow iter + 1$
end while

Example 3.3.3. *Take S, $\boldsymbol{G}^{0/1}$, and $\boldsymbol{y}$ as shown in Example 3.3.2. $\boldsymbol{w}$ is size 4, one weight per row of S. We show S with $\boldsymbol{w}$ after each iteration as an additional column below. IPF iterates over the aggregates in the same order as the rows of $\boldsymbol{G}^{0/1}$.*

Following the pseudo code, we start with $j = 1$. $\boldsymbol{G}^{0/1}[1] = \begin{bmatrix} 1 & 1 & 0 & 1 \end{bmatrix}$ and $\boldsymbol{y}[1] = 5$. These represent the aggregate `date` $= 01$ having a count of 5. As $\boldsymbol{G}^{0/1}[1] \cdot \boldsymbol{w} = 3$, we update $\boldsymbol{w}$ so the first, second, and fourth elements are $5/3$. This is shown in the weight column for $j = 1$ below.

When $j = 2$, $\boldsymbol{G}^{0/1}[2] = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}$ and $\boldsymbol{y}[1] = 5$. As $\boldsymbol{G}^{0/1}[2] \cdot \boldsymbol{w} = 1$, we set the third element of $\boldsymbol{w}$ to be $5/1$, as shown in the column for $j = 2$.

			Weight Values After Iteration					
		iter =	*0*	*1*	*1*	*1*	...	*1*
		j =	*0*	*1*	*2*	*3*	...	*9*
date	**o_st**	**d_st**	$\boldsymbol{w}$	$\boldsymbol{w}$	$\boldsymbol{w}$	$\boldsymbol{w}$	...	$\boldsymbol{w}$
01	*FL*	*FL*	*1*	*1.67*	*1.67*	*1*	...	*1*
01	*FL*	*FL*	*1*	*1.67*	*1.67*	*1*	...	*1*
02	*NC*	*NY*	*1*	*1*	*5*	*5*	...	*3*
01	*NY*	*NC*	*1*	*1.67*	*1.67*	*1.67*	...	*1*

The process continues for all 9 rows of $\boldsymbol{G}^{0/1}$ *until we get the weights after* $j = 9$ *and* $iter = 1$*, shown in the last column. We see that at the end , the weights for the tuples with* `date` $= 01$ *are back to their original value of one. When we go through the process again for* $j = 1$ *and* $iter = 2$*, those weights will be scaled back to* $5/3$*. In this case, IPF will not converge because the sample is missing tuples that fly to and from* `FL`*, but IPF does give us an approximate reweighting.*

We show in Section 3.5 that even when IPF does not converge, the approximate weights still achieve high accuracy for queries asking about tuples in S.

Lastly, while both the reweighting techniques solve for $w(t)$, that linear regression has $m^{0/1}$ parameters while IPF has n_S parameters. Typically, $m^{0/1} < n_S$. Further, since both methods have $\sum_{i=1,B} M_i$ constraints, typically, linear regression is over constrained while IPF is under constrained.

3.3.2 Probabilistic Model Learning

We just presented two different reweighting schemes to debias S using the aggregates Γ. It is important to understand when sample reweighting will fail. For one, the Horvitz-Thompson estimator, which we are approximating by $w(t)$, assumes the support of the sample is the same as the population, i.e., $Pr_S(t) > 0 \; \forall t$. When this does not hold, e.g., when the sampling design is flawed, sample reweighting is inaccurate. Secondly, even if the support is the same,

sample reweighting will fail when tuples exist in P but not in S because the sample will always say those tuples do not exist. This occurs with rare groups and small sample sizes. While we could impute missing rows to S, this risks losing important structural information (S gives us partial information about the manifold P lives on) and slowing down queries.

This section presents our solution to this problem: build a probabilistic model of P using S and Γ and answer queries directly over this model [49, 104]. In order to reason about the population probability distribution, we use the possible world semantics. As existing model learning techniques assume access to P, we present unique modifications and enhancements to adapt these techniques to use Γ and S.

When building a probabilistic model, the first consideration is what class of distributions to use. For example, if the population is believed to be Gaussian in nature, learning a mixture of Gaussians will likely be optimal. As we have no prior knowledge on the population, our main concern is choosing a distribution that can be learned from aggregate data. Similar to [117], we use a Bayesian network (BN) to model the population distribution as a Bayesian network is parameterized by aggregate queries and can scale to many attributes and large data [56]. Unlike [117], which builds the BN from the sample only, the novelty of our BN framework is that is merges S and Γ into BN learning.

BN learning can be broken into two components: structure learning and parameter learning. As we have a biased sample S and aggregates Γ, we need to determine whether to use the sample only (S), the aggregates only (A), or both simultaneously (B) during learning. Intuitively, we will need to use both pieces of information (B) for the best results because while the sample will have more structural information than a set of lower dimensional aggregates, the sample will be missing active domain knowledge (i.e., missing values). Note that while we are specifically interested in the BN structure and parameters, these principle decisions of using samples or aggregates to do *structure learning* and *parameter learning* for a probability distribution are essential for any chosen distribution.

Table 3.3 breaks down the various combinations of learning techniques (these acronyms are used in Section 3.5). As the aggregates may not cover all the attributes, we cannot use

Structure	S	S	B	B
Parameter	S	B	S	B
Method	SS	SB	BS	BB

Table 3.3: Break down of Bayesian network learning techniques and the associated method abbreviation used. S stands for using the sample S only, A for using aggregates only, and B for using both.

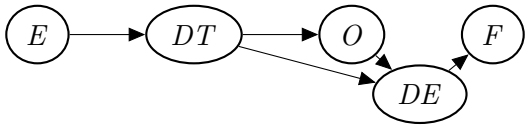


Figure 3.2: Example Bayesian network of flights in the United States (see Table 3.4 for abbreviations).

A for parameter learning. We could use A for structure learning by assuming attributes not covered by the aggregates are uniformly distributed. This uniformity assumption, however, is not optimal. Therefore, we only examine the four methods that do not use just the aggregates for either learning.

Bayesian Networks Overview

We will briefly review Bayesian networks before discussing our learning framework. A Bayesian network is a probabilistic graphical model representing a set of random variables and their conditional dependencies through a directed, acyclic graph. Each edge represents a conditional dependency of the form $\Pr(X_i|Pa(X_i))$ where $Pa(X_i)$ are the parents of node X_i. The example in Figure 3.2 represents a Bayesian network modeling flights in the United States[3]. The joint distribution is

$$\Pr(E, DT, O, DE, F) = \Pr(DT|E)\Pr(O|DT)\Pr(DE|O, DT)\Pr(F|DE)\Pr(E).$$

[3] F (flights date), O (origin), DE (destination), E (elapsed time), DT (distance)

Assuming access to P, there are numerous techniques for learning the BN's structure [32, 92, 56]. Once the Bayesian network structure is known, the parameters $\theta_{i,j,k}$, where $\Pr(X_i = j|Pa(X_i) = k) = \theta_{i,j,k}$, are calculated by minimizing the negative log likelihood of the joint probability subject to constraints enforcing each factor is a (conditional) probability; i.e., for some $Pa(X_i) = k$, $\sum_j \Pr(X_i = j|Pa(X_i) = k) = 1$ and $\Pr(X_i = j|Pa(X_i) = k) \geq 0$. With full access to P, a closed from solution exists that sets the parameters to be the fraction of times each possible state of X_i occurs given the possible states of $Pa(X_i)$ (an aggregate query).

Our challenge is how to do structure learning and parameter learning *without* access to P. We now discuss structure and parameter learning for using both S and Γ (method B). Method S uses the same techniques but assumes Γ is empty.

Learning Network Structure

To exclusively use S and Γ to learn the structure, we adapt the greedy hill-climbing algorithm [59, 92]. The traditional hill-climbing algorithm's goal is to find the structure that maximizes some score. At each step of the algorithm, it makes the "move" the improves the score the most. A "move" is either adding, removing, or reversing a directed edge. If the score cannot be further improved, the algorithm terminates.

We modify the algorithm as follows. To focus on learning from the population before the sample, our algorithm runs in two phases: building from Γ and building from S. As Γ represents ground truth information, we want to build as many edges from Γ before adding edges from S. In the first phase, we make "moves" using Γ until all attributes from Γ are added to the network. Then, if there are any remaining attributes in S not in Γ, we use S to continue building.

In the Γ building phase, we further modify the move selection algorithm to ensure we can score a candidate edge from X_i to X_j. As scoring requires computing a group by query over X_i, X_j, and $Pa(X_i)$, we only consider candidate edges that have the necessary support in Γ; i.e., the attributes X_i, X_j, and $Pa(X_i)$ appear together in some aggregate. In the S

building phase, all edges are allowed.

Our last modification is to "lock in" edges that are added from the Γ phase, meaning we cannot remove them, because we want to keep all structural knowledge from Γ intact. This also prevents overfitting to the sample.

The pseudocode for our greedy hill-climbing algorithm is shown in Algorithm 6. We use the BIC score because it discourages overly complicated structures that could overfit and does not depend on any prior over the parameters [92]. $\mathcal{E}$ is the set of directed edges, $\mathcal{T}$ is the set of conditional probability tables needed to parameterize the network, $\mathcal{P} \in \{1, 2\}$ is the phase of the algorithm, and s represents the BIC score. The functions `CondProbTables` and `BIC` are the same as in the standard algorithm. The function `BuildEdges`, shown in Algorithm 7, determines which moves are allowed; i.e., if an edge has the necessary support.

Note that when the structure learning method is S, we set $\mathcal{P} = 2$ at the beginning.

Learning Network Parameters

We must adapt the standard parameter learning optimization framework to use Γ and S. Inspired by [46, 103] adding parameter sharing and value equality constraints, we add constraints enforcing each aggregate is satisfied. Specifically, let $\mathcal{J}_i = \{j_{i,1}, \ldots, j_{i,d_i}\}$ be the attribute index set of aggregate i; i.e., $\boldsymbol{a}_{i,k} = [a_{k,j_{i,1}}, \ldots, a_{k,j_{i,d_i}}]$. Then, for some $(\boldsymbol{a}_{i',k'}, c_{i',k'}) \in \Gamma_{i'}$ (we use i' and k' to differentiate from the Bayesian network i and k variables), we add the constraint

$$\sum_{\boldsymbol{v} \in \bigtimes_{j' \in \neg\mathcal{J}_{i'}} dom(A_{j'})} \Pr(X_{\mathcal{J}_{i'}} = \boldsymbol{a}_{i',k'}, X_{\neg\mathcal{J}_{i'}} = \boldsymbol{v}) = \frac{c_{i',k'}}{n}$$

where *dom* is the active domain of an attribute, $\bigtimes$ is the cross product, $\neg\mathcal{J}_{i'} = \{1, \ldots, m\} - \mathcal{J}_{i'}$, $X_{\mathcal{J}_{i'}} = \boldsymbol{a}_{i',k'}$ stands for $X_{j_{i',1}} = a_{k',j_{i',1}}, \cdots, X_{j_{i',d_{i'}}} = a_{k',j_{i',d_{i'}}}$, and similarly for $X_{\neg\mathcal{J}_{i'}} = \boldsymbol{v}$. Intuitively, we are summing over all possible values of the attributes, $X_{\neg\mathcal{J}_{i'}}$, that do *not* participate in the aggregate.

Algorithm 6 GreedyHC

$\mathcal{E} \leftarrow \varnothing$, $\mathcal{N} \leftarrow$ all nodes
$s \leftarrow -\infty$, $s' \leftarrow -\infty$, $\mathcal{P} = 1$
do
  if $\mathcal{P} = 1$ **then** $\mathcal{D} \leftarrow \Gamma$
  else $\mathcal{D} \leftarrow S$
  end if
  for $(X_i, X_j) \in \mathcal{N} \times \mathcal{N}$ **do**
    for $\mathcal{E}' \in$ BuildEdges$((X_i, X_j), \mathcal{E}, D, \mathcal{P})$ **do**
      $\mathcal{T}' \leftarrow$ CondProbTables$(\mathcal{D}, \mathcal{E}')$
      $t \leftarrow \text{BIC}(\mathcal{T}', \mathcal{E}')$
      if $t > s'$ **then** $s' \leftarrow t$
      end if
    end for
  end for
  if $attrs(\Gamma) \in \mathcal{E}$ and $|s - s'| \leq 0$ **then** $\mathcal{P} \leftarrow 2$
  end if
while $|s - s'| > 0$
return $\mathcal{E}, \mathcal{T}$

Following Figure 3.2, suppose we know that one aggregate attribute-value pair is that 0.2 percent of flights have $O = \texttt{KA}$, $DE = \texttt{NM}$, and $ET = \texttt{60}$. The added constraint from that aggregate is

$$\sum_{dt \in dom(DT)} \sum_{f \in dom(F)} \theta_{DT,dt,\{\texttt{60}\}} * \theta_{O,\texttt{KA},\{dt\}} * \theta_{DE,\texttt{NM},\{\texttt{KA},dt\}} * \theta_{F,f,\{\texttt{NM}\}} * \theta_{E,\texttt{60},\emptyset} = 0.2$$

We now get the constrained optimization problem in Equation 3.2. $\theta_{i,j,k}$ for a particular i and tuple t means $j = t.A_i$ and $k = \{t.A_{i'} : X_{i'} \in Pa(X_i)\}$. $\boldsymbol{v}$ is the same as before in the

Algorithm 7 BUILDEDGES

if $\mathcal{P} = 2$ and (X_i, X_j) added when $\mathcal{P} = 1$ **then**
  $\mathcal{S} \leftarrow \{\mathcal{E} - (X_i, X_j)\}$
end if
if $\{X_j, X_i, Pa(X_j)\} \in attrs(\mathcal{D})$ **then**
  $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{E} \cup (X_i, X_j)$
end if
if $\{X_j, X_i, Pa(X_i)\} \in attrs(\mathcal{D})$ **then**
  $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{E} - (X_i, X_j) \cup (X_j, X_i)$
end if
return $\mathcal{S}$

sum over all possible values of the attributes that do not participate in the aggregates. $\boldsymbol{v}_{j'}$ stands for the individual value of attribute $A_{j'}$. The $*$ stands for the value of the parent, and it will be set to a value in $\boldsymbol{a}_{i',k'}$ or $\boldsymbol{v}$, depending on if it participates in the aggregate or not.

$$\text{minimize:} \quad -\sum_{t \in S} \sum_{i=1}^{m} \log \theta_{i,j,k} \tag{3.2}$$

$$\text{subject to:} \quad \theta_{i,j,k} \geq 0 \qquad \forall i, j, k$$

$$\sum_{j} \theta_{i,j,k} = 1 \qquad \forall i, k$$

$$\sum_{\boldsymbol{v}} \prod_{j' \in \neg \mathcal{J}_{i'}} \theta_{A_{j'}, \boldsymbol{v}_{j'}, *} \prod_{j \in \mathcal{J}_{i'}} \theta_{A_j, a_{k',j}, *} = \frac{c_{i',k'}}{n} \qquad \forall (\boldsymbol{a}_{i',k'}, c_{i',k'})$$

Note, when our method for *parameter learning* is just S, we do not add the aggregate constraints and can use the closed form solution for learning BN parameters.

Query Answering

Once we have learned the probability distribution of our population, we can answer selection (point) queries probabilistically by calculating $n * \Pr(X_1 = x_1, \ldots, X_m = x_m)$. To answer non-selection queries, such as top-k or group by, we use the BN to generate a sample of data that is representative of the population via forward/logic sampling [117, 81, 68]. Once the sample S' is generated (this is not the original sample S), tuples are uniformly scaled up (i.e., the weight of each tuple is $|P|/|S'|$), and queries are answered as they are for reweighted samples.

3.3.3 Hybrid Query Evaluator

Our hybrid approach integrates the previous two methods into a unified technique for query answering. A naïve approach is to use the BN to reweight the sample. However, this approach does not solve the fact that tuples in the population do not exist in the sample. Our hybrid approach needs to rely on the sample only when a tuple exists in the sample.

Our hybrid approach is straightforward. When a point query gets issued, if the tuple being queried is in the sample, it uses the reweighted sample. Otherwise, it uses the BN. If the user issues a top-k query, it gets sent to the reweighted sample as the sample is more likely to contain heavy hitters, i.e., top-k values. In both cases, if the support of the sample is known to be different from the support of the population, the BN should always be used. It is future work to detect if the support is sufficient to use the reweighted sample.

The motivation of this demarcation between using the sample versus the Bayesian network is due to the inherent problems with sample reweighting. If the tuple does not exist or the support is not sufficient, the sample achieves poor accuracy. We are simply capturing this failure in our query evaluator by only using the sample when we believe it will achieve the lowest error.

3.4 Optimization

We showed how to incorporate aggregates into sample reweighting and BN learning. The main challenges to efficient implementation are that every individual aggregate $(\boldsymbol{a}_{i,k}, c_{i,k})$ adds complexity to the optimization, e.g., each new aggregate is one more iteration of weight rescaling in IPF, and more importantly, the constrained optimization in Equation 3.2 is a nonlinear optimization with $O(\prod_{j \in \neg \mathcal{J}_i} N_j)$ variables for *each* $(\boldsymbol{a}_{i,k}, c_{i,k})$. The large number of variables along with the added complexity of using nonlinear constraints [115] makes solving computationally expensive. Further, our constraints prevent us from optimizing each BN factor independently, a benefit of using traditional BNs.

To solve these problems, we present two optimization techniques: pruning the least informative aggregates and simplifying our constraint optimization.

3.4.1 Aggregate Selection

Our goal is to reduce the number of aggregates, i.e., $|\Gamma|$, *before* using them in reweighting and Bayesian network learning. The natural choice is to choose the B most informative aggregates; i.e., the B aggregates that minimize the distance between the true distribution and some distribution parametrized by the aggregates (more on this later). We chose to minimize the Kullback-Leibler (KL) divergence, which measures the amount of information lost between two distributions, because of the existing work on minimizing the KL divergence (e.g., Chow-Liu trees [39] and VAEs [51]).

To minimize the KL divergence, we assume, like BNs, that our approximate distribution is a product distribution because there is a known product distribution that minimizes the KL divergence from the true distribution: a Chow-Liu tree [39] (a second-order product approximate) and its higher-order extension, a k-order t-cherry junction tree [31, 118].

Before defining a *k-order t-cherry junction tree*, recall that a *junction tree* [124, 109] is a tree structure where a node (cluster) is defined by a subset of random variables, denoted $\boldsymbol{X}_C$, and is associated with the distribution $\Pr(\boldsymbol{X}_C)$. Every tree edge is called a separator

and defined by the subset of random variables, denoted $\boldsymbol{X}_S$, contained in the intersection of the two clusters being linked. A junction tree must also satisfy the running intersection property that all tree nodes containing the variable X must form a connected region and have every random variable contained in some cluster.

A *k-order t-cherry junction tree* is a junction tree with the added properties that (1) every cluster (tree node) consists of exactly k random variables, and (2) every separator consists of exactly $k-1$ random variables. Assuming access to P, the greedy algorithm for building a t-cherry tree is to score all possible cluster-separator pairs by $I(\boldsymbol{X}_C) - I(\boldsymbol{X}_S)$ where $I(\boldsymbol{X}_C) = \sum_{i \in C} H(X_i) - H(\boldsymbol{X}_C)$ is information content and H is entropy. It greedily adds new cluster-separator pairs with the highest scores as long as, at each iteration, a new random variable is being covered and the separator to be added is contained in an already added cluster. The algorithm terminates once all random variables are covered.

As we do not have P, we must modify the k-order t-cherry junction tree algorithm. Our pseudocode is shown in Algorithm 8 (detailed algorithm is shown in Appendix A) where $\mathcal{C}$ is the set of potential clusters to add and $\mathcal{C}'$ is the set of chosen clusters. As our algorithm does not have access to P, our first modification is that the `GenClusterSeparatorPairs` function only initializes cluster-separator pairs that have support in Γ; i.e., the attributes of the cluster are contained in some aggregate. This allows us to calculate the cluster-separate pair score from Γ alone.

For our second modification, as our aggregate budget B may larger than the number of attributes, once the algorithm terminates, we restart it, building a new tree from scratch. To avoid creating duplicate clusters, we disallow previous cluster-separator pairs from being added. This is shown in line 8 by subtracting $\mathcal{C}'$ from the initial set of clusters. Once our algorithm generates B clusters, we filter Γ so that each $\boldsymbol{\gamma}_i$ must be equal to the attributes associated with one of the clusters.

Algorithm 8 Modified t-cherry tree.

```
C ← sorted(GenClusterSeparatorPairs)
C' ← { highest scored cluster-separator in C }
while |C'| < B do
    for c ∈ C do
        if c's separator contained in some cluster ∈ C'
         and new attribute covered by c then
            add c to C'
        end if
    end for
    if all attributes covered then
        start new tree
        C ← sorted(GenClusterSeparatorPairs - C')
    end if
end while
return C'
```

3.4.2 Bayesian Network Simplification

To optimize our constrained optimization, we want each BN factor $\Pr(X_i|Pa(X_i))$ to be optimized independently with *linear* constraints. To do this, we enforce a topological solving order and limit the aggregates added to our model.

To understand how to do this, we first need to discuss what a topological solving order is. For now, assume we can solve each BN factor independently. A topological solving order is simply solving for the factors in topological order, meaning every parent node is optimized before its child node. For example, if $X_{i'}$ is a parent of X_i, then we solve for $\theta_{i',j,k}$ before solving for $\theta_{i,j,k}$. Once $\theta_{i',j,k}$ is solved, denote it $\overline{\theta}_{i',j,k}$.

To enforce linear constraints and independent solving, we restrict our model to only add

aggregate constraints that act on single factors, i.e., aggregate constraints over a child node X_i and its parents if the parents are in the aggregates [4]. This means for child node X_i, the constraints in Equation 3.2 will only contain the the product of the child parameter $\theta_{i,j,k}$ with its ancestors because the other factors have marginalized out. By itself, this has only reduced the number of product factors. The key is topological solving order. By insuring that the parents are solved for before the children, at the time of solving for $\theta_{i,j,k}$ for a particular X_i, the ancestor terms are already known and become a constant in the constraint, meaning the $\theta_{i,j,k}$ for X_i are the only parameters. By removing aggregate constraints that act on multiple different BN factors, we can turn our nonlinear constraints into linear ones.

This linearity of constraints for a particular factor allows us solve for factors independently. As we only include constraints on single factors and only those factor's parameters are unknown, we can solve factors independently.

Example 3.4.1. *Take the example network from Figure 3.2. Suppose we have two aggregates over E and (O, DE). A topological ordering of all nodes is E, DT, O, DE, F. To simplify indexing, instead of using i', k' to index the aggregates, we will use the associated values as indexes for the counts. For example, if one aggregate of (O, DE) is $(\boldsymbol{a}_{2,k'}, c_{2,k'}) = ([\texttt{WI}, \texttt{MN}], 10)$, we write this as $c_{2,\{\texttt{WI},\texttt{MN}\}} = 10$ (2 representing the second aggregate).*[5]

We solve for E first by solving

$$
\begin{aligned}
&\textit{minimize:} && -\sum_{t \in S} \log \theta_{E,j,\emptyset} && \\
&\textit{subject to:} && \theta_{E,j,\emptyset} \geq 0 && \forall j \\
& && \sum_{j} \theta_{E,j,\emptyset} = 1 && \\
& && \theta_{E,j,\emptyset} = c_{1,j}/n && \forall j \in dom(E).
\end{aligned}
$$

[4] If we have an aggregate over X_i and some other $X_{i'}$, we can turn that aggregate into one over just X_i by grouping by X_i, $Pa(X_i)$ and summing the counts.

[5] Even though both values are US States, the set construction is valid because they are really origin state WI and destination state MN.

Note that because of marginalization of the BN factors, the aggregate constraints do not sum over all possible values of (DT, O, DE, F)*. Once this optimization is solved, we have* $\overline{\theta}_{E,j,\emptyset}$*. We solve our next node,* DT*, in closed form because we have no constraints over* DT*. We solve* O *next by*

$$
\begin{aligned}
&\textit{minimize:} && -\sum_{t \in S} \log \theta_{O,j,k} && \\
&\textit{subject to:} && \theta_{O,j,k} \geq 0 && \forall j, k \\
& && \sum_{j} \theta_{O,j,k} = 1 && \forall k \\
& && \sum_{k \in dom(DT)} \theta_{O,j,k} \sum_{\ell \in dom(E)} \overline{\theta}_{E,\ell,\emptyset} \overline{\theta}_{DT,k,\ell} = \sum_{v \in dom(DE)} \frac{c_{2,\{j,v\}}}{n} && \forall j \in dom(O).
\end{aligned}
$$

Note we turn the constraint over (O, DE) *to be one just over* O *by aggregation. We use* (O, DE) *again when solving for* DE*.* F *is solved in closed form.*

We can further improve efficiency in two ways. First, as we can solve factors independently, if there is a factor without any aggregate constraint, we can directly solve for the parameters in the traditional manner through an aggregate query over the sample. Second, we can limit the number of parent nodes each child can have in Bayesian network structure learning by modifying the hill-climbing algorithm to prevent adding/reversing edges if a child already has enough parents. By limiting this number, we are making the summation in the constraint solver smaller and thus improving efficiency.

3.5 Evaluation

In this section, we evaluate the query accuracy and query execution time of THEMIS. We compare THEMIS's hybrid approach to always using the optimal sample reweighting technique and always using the best Bayesian network technique. We further investigate the performance of the two different sample reweighting techniques (linear regression and IPF)

and of Bayesian network learning technique variations. Lastly, we demonstrate the benefits of using our pruning technique. We do not plot results for our constraint solver optimization from Section 3.4.2 as experiments did not finish in under 10 hours without using the optimization.

3.5.1 Implementation

We implemented THEMIS in three parts: the sample reweighter, the BN learner, and the query evaluator. The linear regression, IPF, and BN constraint solving were implemented in Python 3.7 using Numpy, Scipy, and Pandas. Linear regression was solved using Scipy's nnls solver to enforce the linear regression weights being positive. IPF was custom implemented using Pandas. The BN constraint solving was handled in Python version 3.7 using Scipy's optimize package, specifically the minimize function with the "trust-const" method. We used the default tolerances of $1 * 10^{-8}$ and ran the algorithm for at most 100 iterations. Since the constraint solving is approximate, occasionally a model parameter would be set to some very small negative number. We set these parameters to zero.

The BN structure learning and inference were implemented in R 3.2 using the BNLearn and gRain package (gRain for exact inference). For top-k queries, we created samples that were the same size as input sample. Lastly, we limited our Bayesian networks to have at most two parents because population aggregates are rarely more than three-dimensional. Further, to avoid the summation over active domain values for the BN constraint solver to become too large and slow down solving time for methods BB and SB (see Table 3.3 and Section 3.4.2), we restrict our network for methods BB and SB so that if a child node is contained in some aggregate and has more than one parent, then the child node and its parents must be all contained in one aggregate (meaning we get direct equality constraints). This implies we will only be summing over at most one active domain.

After learning, the samples with weights stored as an additional column were stored and queried in a Postgres 9.5 database. We performed all experiments on a 64bit Linux machine running Ubuntu 16.04.5. The machine has 120 CPUs and 1 TB of memory. The Postgres

Attribute	Shorthand	N_i
fl_date	FD	12
origin_state	OS	51
dest_state	DS	51
elapsed_time	ET	32
distance	DT	75

Table 3.4: Flights data attributes and active domain size.

Attribute	Shorthand	N_i
movie_title	MT	62445
movie_year	MY	13
movie_country	MC	3
name	N	48395
gender	G	2
rating	RG	10
top_250_rank	TR	251
runtime	RT	91

Table 3.5: IMDB data attributes and active domain size.

database also resides on this machine with a shared buffer size of 250 GB.

3.5.2 Datasets

We use a flights dataset [5] (all United States flights in 2005 with $n = 6,992,839$), an IMDB dataset [83] (actor-movie pairs released in the United States, Great Britain, and Canada with $n = 846,380$), and a synthetic CHILD Bayesian network dataset generated using BNLearn [6] with $n = 20,000$. We preprocess the datasets to remove null values and bucketize the real-valued attributes into equi-width buckets. For the two real-world datasets, the attributes and attribute abbreviates are shown in Table 3.4 and Table 3.5.

We take three samples from Flights: uniform (Unif), flight month of June (June), and flights leaving from a four corner state of CA, NY, FL, WA (SCorners). The S stands for the supported Corners sample. Each are 10 percent samples with a 90 percent bias, meaning 90

d	B	Flights	IMDB
2	1	E & DT	MY & RT
	2	DE & DT	RG & RT
	3	O & DT	MY & MC
	4	F & DE	MY & G
3	1	O & DE & DT	MY & RG & RT
	2	O & E & DT	MY & MC & RT
	3	F & E & DT	MY & G & RT
	4	DE & E & DT	MC & RG & RT

Table 3.6: The 4 2D and 3D Flights and 4 2D and 3D IMDB data aggregate attributes chosen by the pruning technique.

percent of the rows are from the selection criteria. We also take a corner states 10 percent sample with 100 percent bias (Corners).

We likewise take three samples from IMDB: uniform (Unif), movie country of Great Britain (GB), and movies with ratings 1, 5, or 9 (SR159). We similarly give these 10 percent samples a 90 percent bias and take a 10 percent sample with 100 percent bias of the ratings sample (R159).

As the CHILD data is used to examine our pruning technique, we just use a 10 percent uniform sample.

3.5.3 Experimental Setup

As real population reports typically have aggregates of one, two, or three dimensions (e.g., Excel tables), we use $d = 1$, 2, or 3. We prune all possible aggregates by our pruning technique to produce from $B = 1$ to 4 aggregates. Table 3.6 shows the aggregates chosen. For IMDB, we only consider aggregates from the attributes MY, MC, G, RG, RT to investigate the impact of having aggregates that do no cover all attributes.

To measure accuracy, we run top-k queries for $K = 30$ and point queries where the point query selection values are selected from the population's light hitters (smallest values), heavy hitters, and randomly. We run 100 point queries for each of the three selections, for a total of 300 point queries per attribute set.

For Flights, we issue point queries over all possible attribute sets of size two to five (total of 26). We run top-k queries for all possible two and three attribute set combinations (total of 20). For IMDB, as there are too many attributes to run all possible point queries, we randomly choose 20 three dimensional attribute sets to run our point and top-k queries over. Note we look at all attributes for queries, not just those we have aggregates over.

Following [129], we measure precision and recall of the top-k queries and report the F measure. For point queries, we use the error metric of percent difference, $2 * |true\_value - est\_value|/|true\_value + est\_value|$. We use percent difference rather than percent error to avoid over emphasizing errors where the true value is small and to ensure missed groups get the maximum error of 200 percent.

Lastly, when measuring runtime (Section 3.5.9), as all reweighted samples are stored and accessed the same, we only look at the runtime for one reweighted sample.

3.5.4 Overall Accuracy

We first evaluate which method is optimal for answering queries over different biased samples. Using $d = 2$ and $B = 4$, we compare THEMIS's hybrid approach (pink) to the best linear reweighting technique of IPF (orange) and to the best Bayesian network technique of BB (blue) (BB means it uses both Γ and S to learn the BN). We further compare against standard uniform reweighting as a baseline (red).

Figure 3.3 and Figure 3.4 show boxplots of the percent difference of 100 heavy and 100 light hitter point queries across the samples. The median value is the black line and the average is the black X. For reference, Table 3.7 shows the percent improvement of the 25th, 50th, and 75th percentiles of THEMIS's hybrid approach to uniform reweighting for Flights.

We see that for the samples that have the same support as the population (first three), THEMIS's hybrid technique achieves the lowest error. For the Flights sample without support (Corners), the BN technique (BB) performs best, but hybrid performs better that IPF, indicating that hybrid mitigates the problem of mismatching support. BB does not

Hitters	**Percentile**	Unif	June	SCorners	Corners
Heavy	25	4.2	13.6	168.3	6.1
	50	1.8	69.7	61.9	2.7
	75	1.4	29.6	34.4	2.2
Light	25	∞	∞	∞	45
	50	1.7	1.7	1.7	1.4
	75	1.0	1.0	1.0	1.0

Table 3.7: Percent improvement of percentiles for hybrid compared to Unif for the queries from Figure 3.3. The infinite value represents that hybrid has zero error.

perform optimally for the `IMDB` sample without support (R159) because of queries over the very dense attribute `N` (48,000 distinct values). BB learns that `N` is uniformly distributed and underestimates queries over `N` because all values are equally likely. This makes BB perform worse than IPF. Further, as three attributes are not covered by aggregates, one of them (top 250 rank/`TR`) being highly correlated with the attribute biasing the sample, BB and IPF overfit to the sample and receive high error for queries over `TR`.

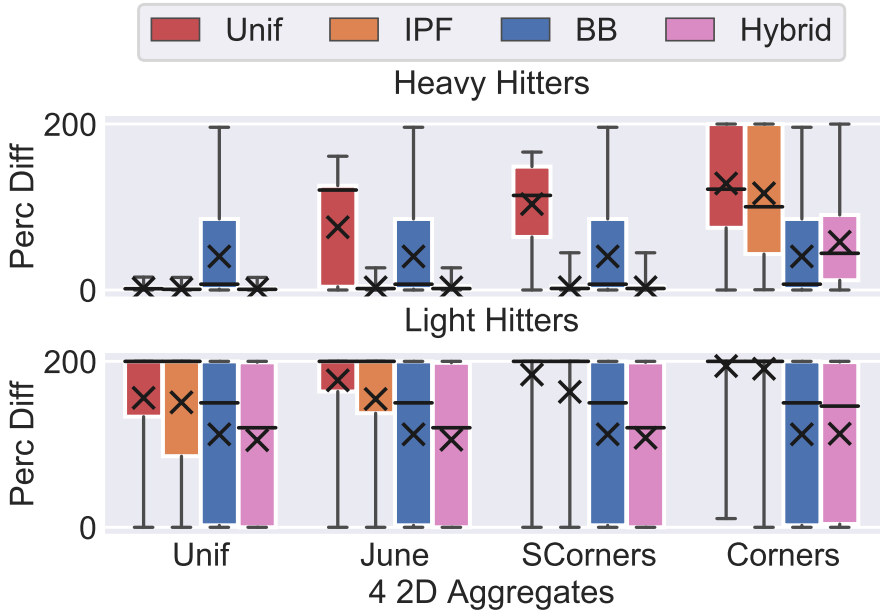


Figure 3.3: 100 heavy and light hitter point query percent difference for `Flights` biased samples ($d = 2$, $B = 4$).

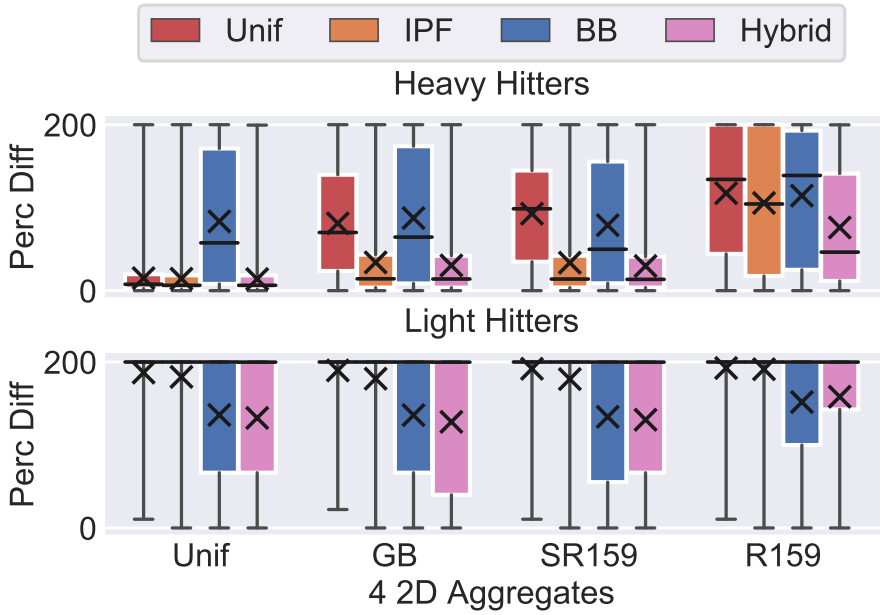


Figure 3.4: 100 heavy and light hitter point query percent difference for `IMDB` biased samples ($d = 2$, $B = 4$).

Figure 3.5 shows the error of 100 random point queries for `Flights` grouped by if the query was in the sample or not and Figure 3.6 shows the same for `IMDB`. The separation mimics THEMIS's hybrid query evaluator decision, which is why hybrid receives the same error as IPF for queries in the sample and as BB for queries not in the sample. We see the same trend that IPF only performs well when the support is the same and that BB performs best when the support is not the same (Corners and R159). Unlike heavy hitter queries, we see that BB achieves lower error than IPF for `IMDB` because random point queries with the attribute `N` that are in the sample have smaller count values. BB still underestimates queries over `N`, but the underestimation is less severe for random point queries than heavy hitter. Therefore, its error is lower.

We now evaluate which method is optimal for answering top-k queries using `Flights` and `IMDB`. We dive more deeply into top-k performance by varying the number of 2D aggregates after adding 5 1D aggregates, one for each attribute. We do not show results for other K values as the trends are similar.

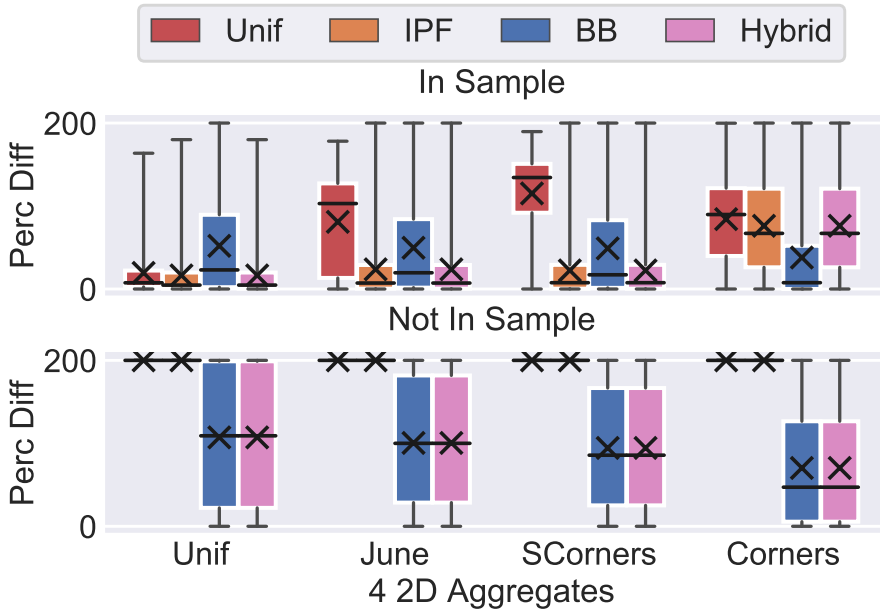


Figure 3.5: 100 random point query percent difference for `Flights` biased samples ($d = 2$, $B = 4$).

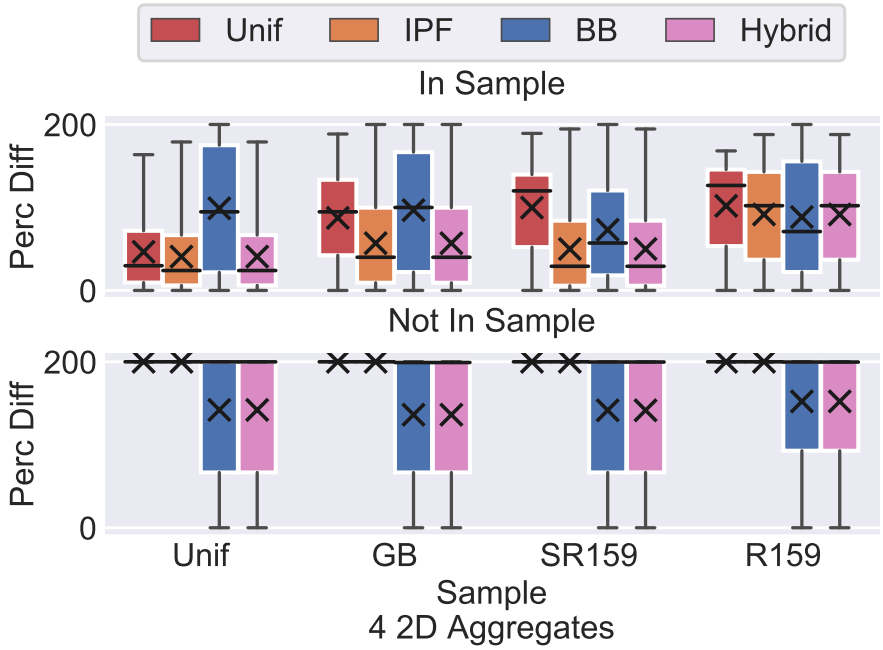


Figure 3.6: 100 random point query percent difference for `IMDB` biased samples ($d = 2$, $B = 4$).

Figure 3.7 (Flights) shows a dramatic improvement in the IPF/hybrid approach (they are the same for top-k) for SCorners compared to Corners, which supports the results shown for point queries. A similar trend is shown in Figure 3.8 (IMDB) for SR159 and R159. Further, BB improves the most from adding more 2D aggregates as it is learning more structural and parameter information from the population.

Lastly, all methods for the IMDB top-k queries score on average 17 percent lower on F measure than Flights top-k queries. This is due to both the larger number of attributes and larger active domain size compared to sample size (fewer top-k tuples are going to be included in the sample) and the fact that all attributes are not covered by aggregates.

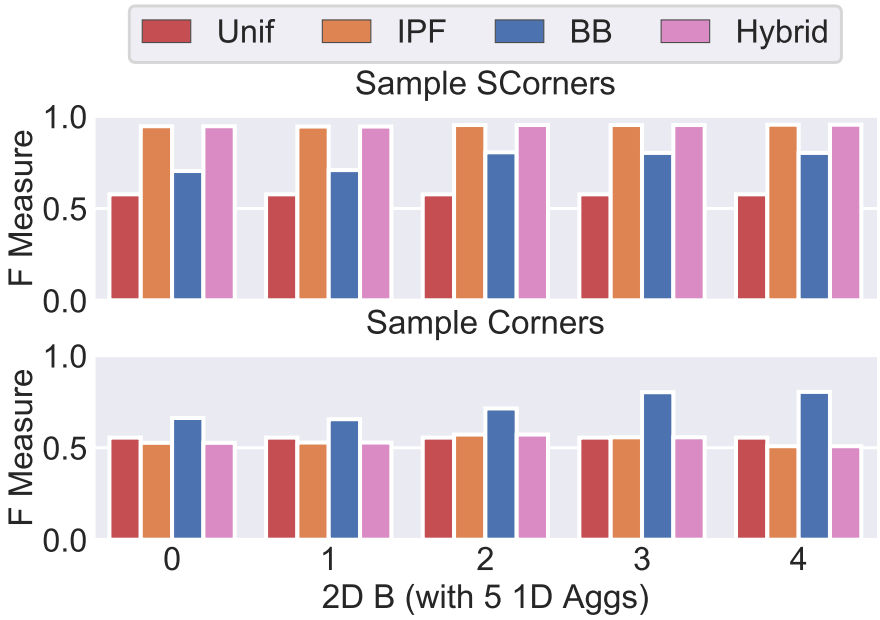


Figure 3.7: F measure of top 30 for 5 1D aggregates and varying numbers of 2D aggregates for Flights.

Overall, we see that THEMIS's hybrid technique outperforms the alternatives for point queries and top-k queries and mitigates the problem of the sample and population having different support.

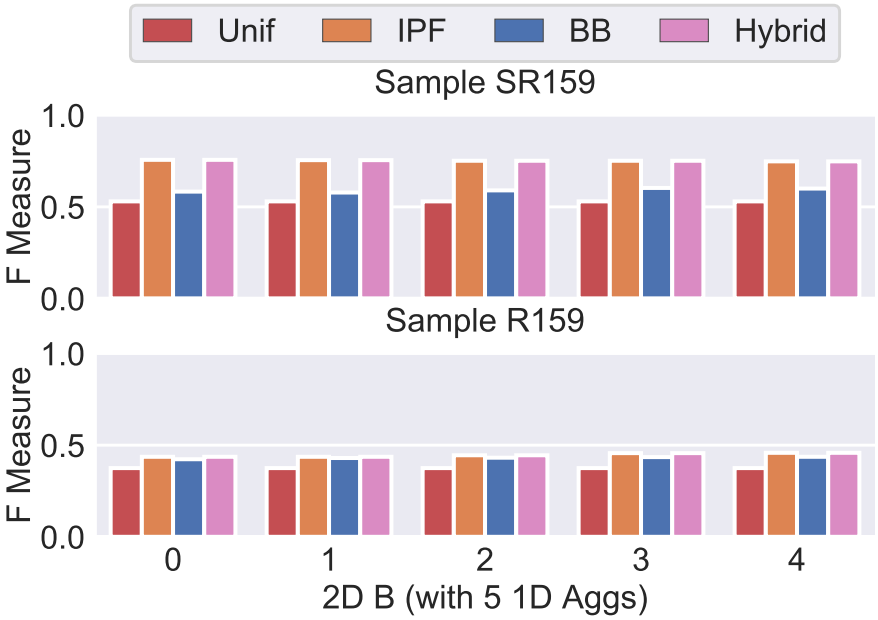


Figure 3.8: F measure of top 30 for 5 1D aggregates and varying numbers of 2D aggregates for IMDB.

3.5.5 Changing Aggregate Knowledge

To examine how varying our aggregates impacts accuracy, we run 100 random point queries on two Flights samples and two IMDB samples as we add 1D, 2D, and 3D aggregates. To show the impact of attribute coverage, we add the 1D aggregates in two different orders. For Flights, we add the 1D aggregates in order A—F, O, DE, E, DT—and order B, the reverse of order A. For IMDB, we add the 1D aggregates in order A—MY, MC, G, RG, RT—and order B, the reverse of order A. For both datasets, after adding the 1D aggregates, the 2D and 3D aggregates are added as in Table 3.6.

Figure 3.9 shows a line plot (to better show trends) of the average percent difference for SCorners (top row) and June (bottom row) for order A (left column) and order B (right column). For SCorners, the largest improvement in all THEMIS methods (IPF, BB, and hybrid) is when adding the second attribute in order A (fourth attribute in order B). This is O (the attribute SCorners is biased on) which indicates that THEMIS is correctly learning which attribute is causing bias. The result is replicated with the June sample and attribute

F, with the IMDB sample of SR159 with attribute RG, and with the IMDB sample of GB with attribute MC. Although, the improvement is less pronounced with IMDB as we do not have covering aggregates and the active domain is larger. Figure 3.11 shows the average percent

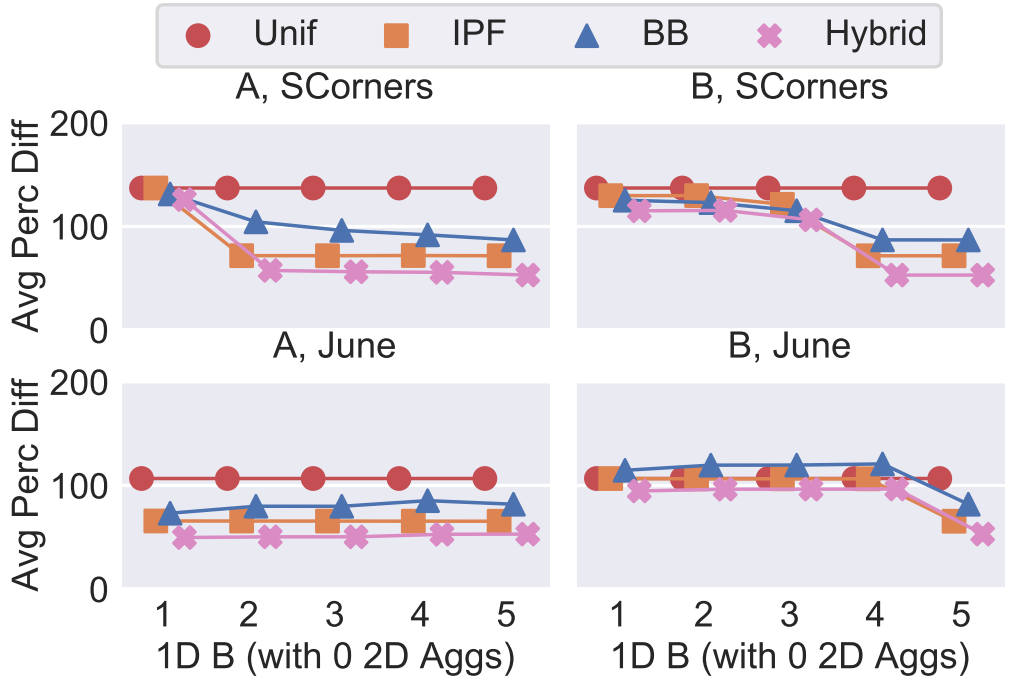


Figure 3.9: Average percent difference of 100 random point queries for SCorners and June for Flights as more 1D aggregates are added.

difference for the same two samples as 2D aggregates are added for Flights. Figure 3.12 shows the same for IMDB. We see that BB improves the most with more aggregates. However, we see diminishing returns after adding 2 aggregates. As more aggregates are added, BB gets closer to hybrid while IPF does not significantly improve with more 2D aggregates. This indicates that when enough population data is added, knowledge from the sample is overridden by the population data.

Figure 3.13 shows the average percent difference for the samples SCorners and Corners as 3D aggregates are added after adding 5 1D aggregates for Flights. Figure 3.14 shows the same for IMDB with the samples R159 and SR159. A horizontal green line is added

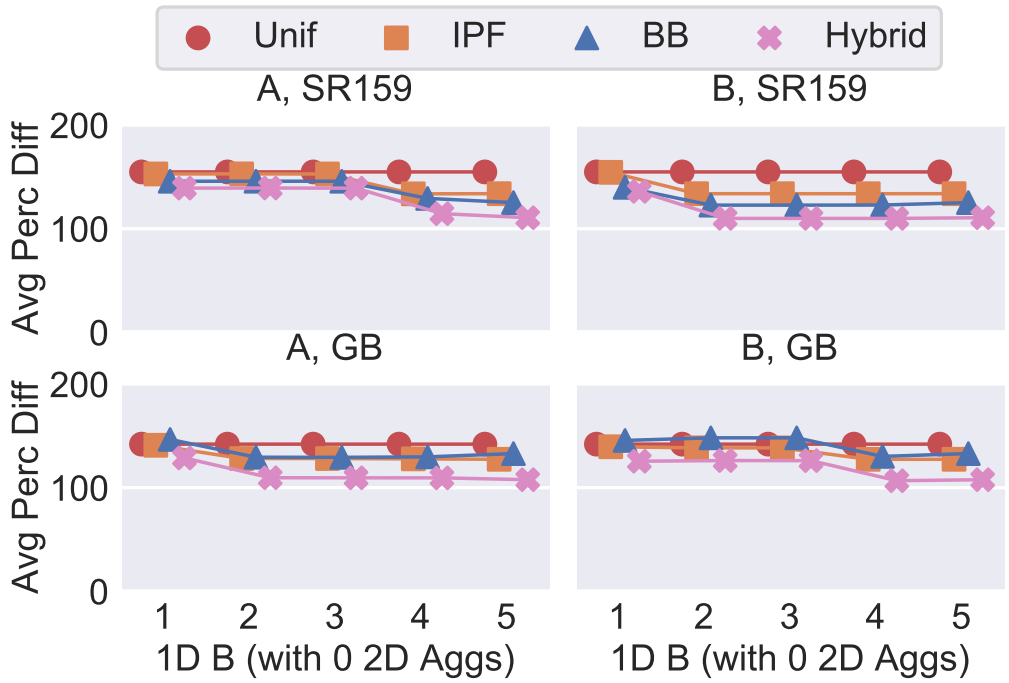


Figure 3.10: Average percent difference of 100 random point queries for SR159 and GB for `IMDB` as more 1D aggregates are added.

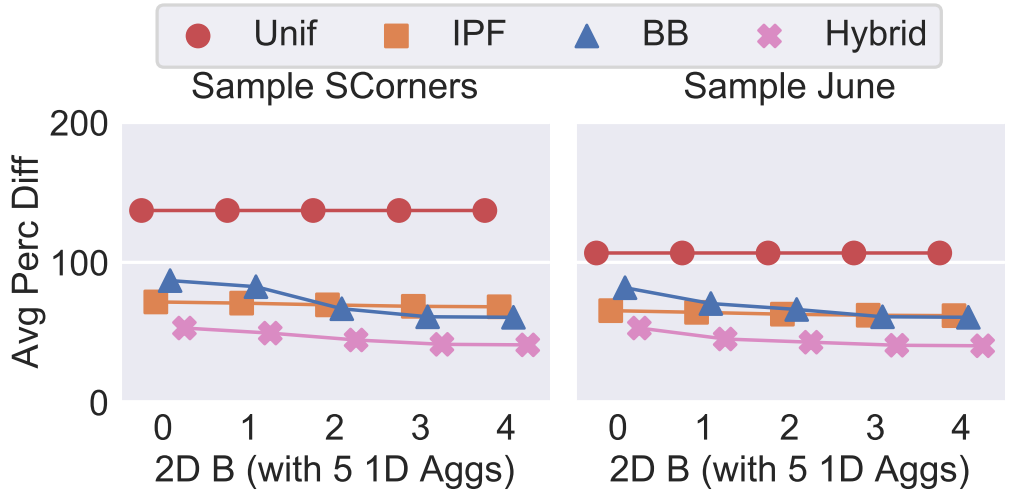


Figure 3.11: Average percent difference of 100 random point queries for SCorners and June for `Flights` as more 2D aggregates are added after adding the 5 1D aggregates from Figure 3.9.

indicating the average percent difference for hybrid after 4 2D aggregates were added.

We see the same trend that aggregates improve BB more than sample reweighting. We

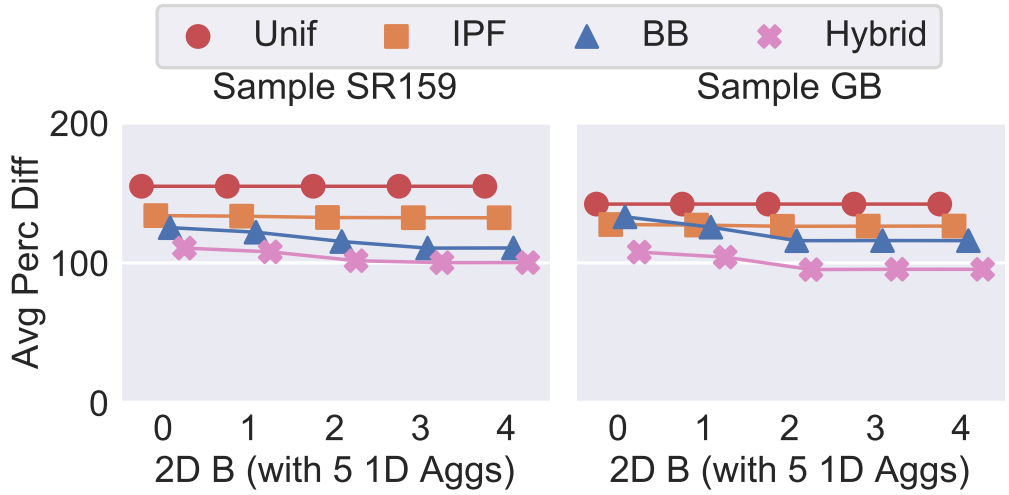


Figure 3.12: Average percent difference of 100 random point queries for SR159 and GB for IMDB as more 2D aggregates are added after adding the 5 1D aggregates from Figure 3.10.

further see that adding 3D aggregates causes faster convergence. After adding just a single 3D aggregate are we able to achieve the same error as adding 4 2D aggregates for SCorners.

When looking at IMDB, we notice that adding aggregates does not significantly improve any method. This is due do lacking attribute coverage in the aggregates, especially over dense attributes like N that cause significant error in query answers.

Flights, on the other hand, does have attribute coverage. This means adding 3D aggregates can cause BB and hybrid to have lower error than having 4 2D aggregates, as is shown with June have slightly lower error with 4 3D aggregates than 4 2D aggregates. This trend does not hold with SCorners due to BB learning a less optimal network structure after adding 2 3D aggregates (shown by the dip in the blue line at 1 3D aggregate). With one 3D aggregate only, BB learns that O (origin) should be conditioned on DE (destination) and DT (distance). This makes sense because where a flights starts and how far it goes determines the possible destinations. With more aggregates, however, the relationship learned is that DE is conditioned on DT and O is conditioned on DT. While similar, this relationship does not accurately represent the underlying distribution as much as the former structure. As our structure learning algorithm is approximate, it will not always learn the optimal structure.

The overall trend is that adding 3D aggregates can improve convergence rates, but does

not significantly improve the error for hybrid over just using 2D aggregates.

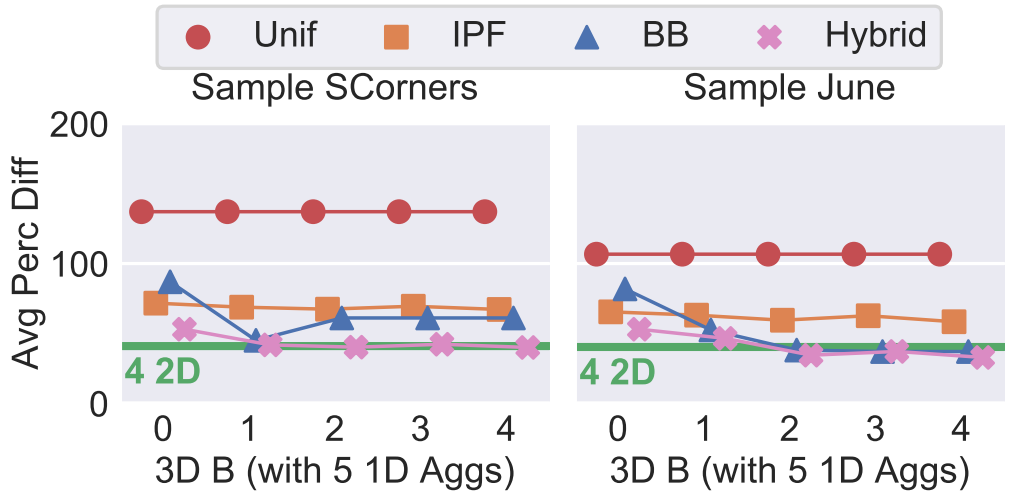


Figure 3.13: Average percent difference of 100 random point queries for SCorners and Corners for `Flights` as more 3D aggregates are added after adding the 5 1D aggregates from Figure 3.9.

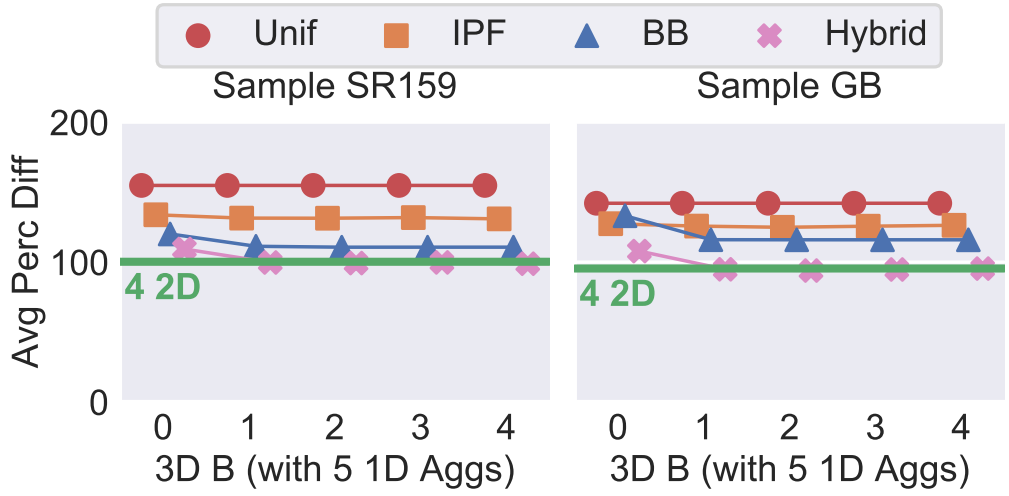


Figure 3.14: Average percent difference of 100 random point queries for SR159 and R159 for `IMDB` as more 3D aggregates are added after adding the 5 1D aggregates from Figure 3.10.

Lastly, to examine how the amount of sample bias impacts accuracy, we measure the average percent difference for 100 random point queries using 4 2D aggregates on the `Flights`

sample of Corners as we decrease the percent bias from 100 percent (Corners sample) to 90 percent (SCorners sample), shown in Figure 3.15. As soon as the support is the same (bias < 100), sample reweighting techniques start performing significantly better. THEMIS's hybrid approach is able to mitigate this difference and performs better than IPF for 100 percent bias.

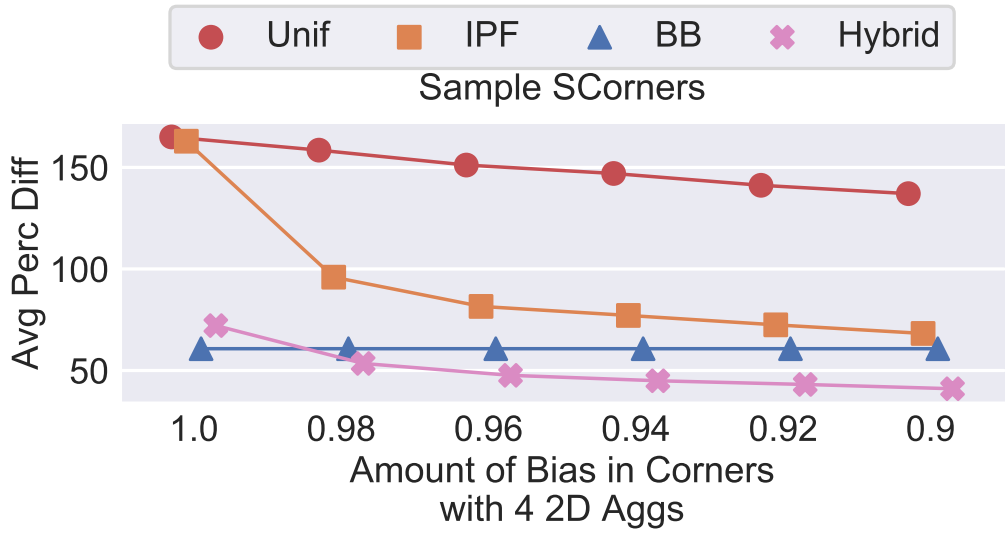


Figure 3.15: Average percent difference of 100 random point queries for the SCorners using 4 2D aggregates as we decrease the percent bias.

3.5.6 Bayesian Network Performance

We now dive into the different BN approaches for structure and parameter learning. Recall that when learning a BN, we can use only the sample (S) or both the sample and aggregates (B) (see Table 3.3). Since the aggregate information (A) is not always covering, we cannot use it exclusively to learn the parameters. Further, is it suboptimal to learn the structure using A because we must assume non-covered attributes are uniformly distributed. However, for comparison, we do examine using just A for structure learning and B for parameter learning. Therefore, the approaches we compare are SS (light green), BS (dark green), SB (light blue), BB (dark blue), and AB (grey).

Using these methods, we measure average percent difference on 100 heavy and light hitter point queries on SCorners while increasing the number of 2D aggregates after adding 5 1D aggregates. The results are in Figure 3.16. We see that all approaches perform better for heavy hitters than light hitters and that BB performs best overall. The only time this does not hold is for heavy hitter queries on SCorners once three 2D aggregates have been added. Note that BB is optimal for light hitter queries once three 2D aggregates have been added. This is due to BB learning a different network structure after adding the aggregate over `DT` and `OS`. It adds an edge from `DT` to `OS` with three aggregates when is had an edge from `DE` to `OS` with only two aggregates. However, the error improvement for light hitter queries for BB after adding three 2D aggregates outweighs the error decrease for heavy hitter queries. We also see that as the number of aggregates increases, AB converges to BB because the population information overrides any sample information given for `Flights`.

Another important trend is that using the sample versus both the aggregates and the sample is more important for parameter learning than for structure learning. We see little difference in accuracy between SS and BS, but SB and BB outperform them both. BB is only slightly more accurate than SB for light hitter queries after adding three aggregates.

3.5.7 Sample Reweighting Performance

We now compare the two different sample reweighting techniques of linear regression (LinReg) and IPF. As we do not see a drastic improvement in adding 2D aggregates when doing sample reweighting, we focus on comparing how they perform on the different `Flights` samples by measuring error on 100 random point queries using 4 2D aggregates.

Figure 3.17 shows boxplots of the percent difference of LinReg, IPF, and Unif. We see clearly that IPF outperforms LinReg on all cases. While LinReg does outperform Unif in all biased samples (not the uniform sample), it does not outperform IPF due to the correlations that exist in the data. For example, to satisfy aggregates on the `DT` attribute, LinReg will add weight to the highly correlated attribute values of `E`. This will help satisfy aggregates on `DT` but will overall hurt performance because any other tuple with the correlated attribute

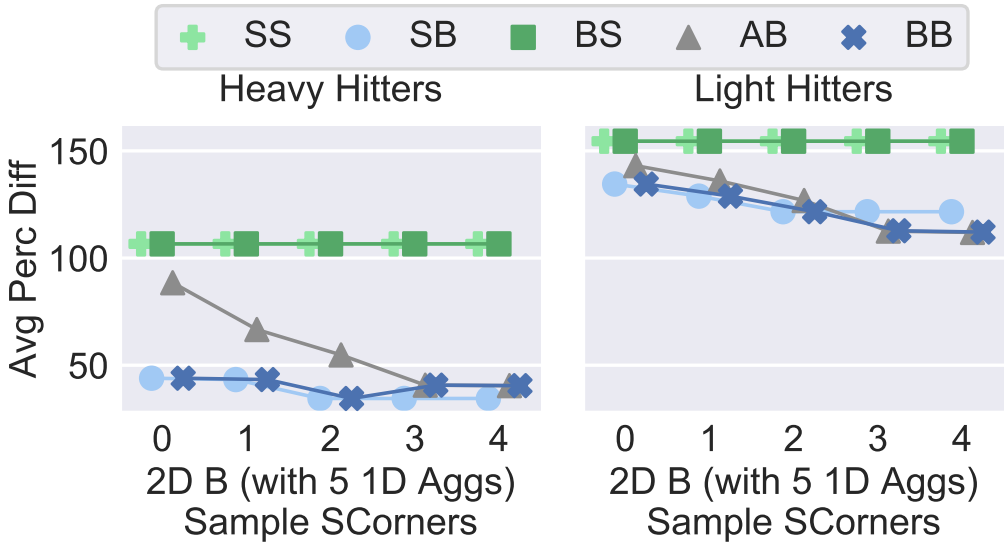


Figure 3.16: Average percent difference of 100 heavy hitter and light hitter queries over SCorners for the 5 different Bayesian techniques as more 2D aggregates are added with 5 1D aggregates already included.

values will have an incorrect weight.

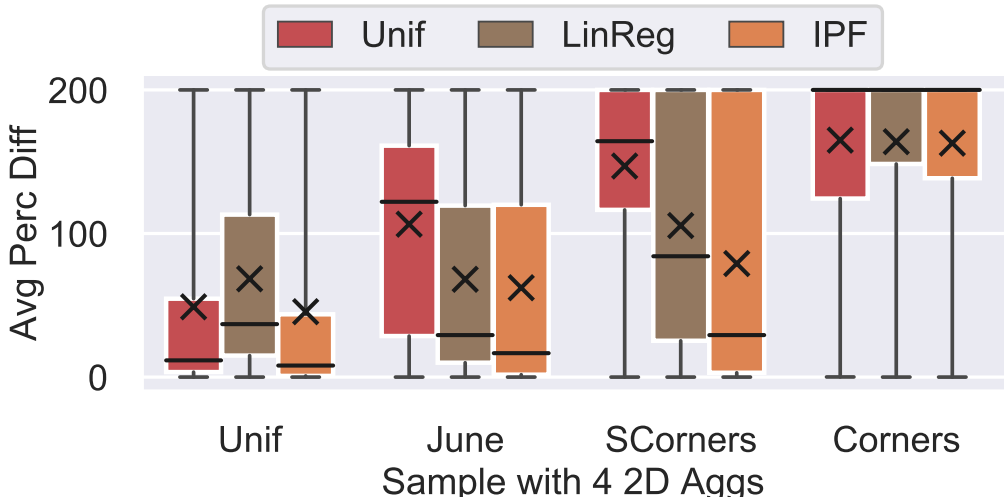


Figure 3.17: Percent difference of 100 random point queries over four different `Flights` samples with 4 2D aggregates.

3.5.8 Pruning Effectiveness

We use the BNLearn CHILD dataset to examine our pruning technique for choosing aggregates. We use the two Bayesian approaches of BB and AB and measure average percent difference for 100 random point queries for 10 randomly chosen attribute sets of size 2, 4, 6, 8, and 10 for a total of 50 attributes sets and 5,000 total point queries. We compare the techniques as we add from 5 to 65 2D aggregates using our pruning technique (Prune) and randomly (Rand) after adding full 1D aggregates. In addition to comparing the error, we also plot the average percent difference if the true Bayesian network is known (optimal error).

Figure 3.18 shows the average percent difference where the dark colors are for Prune while the light colors are for Rand. The red line shows the median optimal error. We see, again, that BB performs better the AB, especially when fewer aggregates are added. Further, the error with Rand improves more slowly than using Prune to add aggregates. If enough aggregates are added, however, the techniques eventually converge.

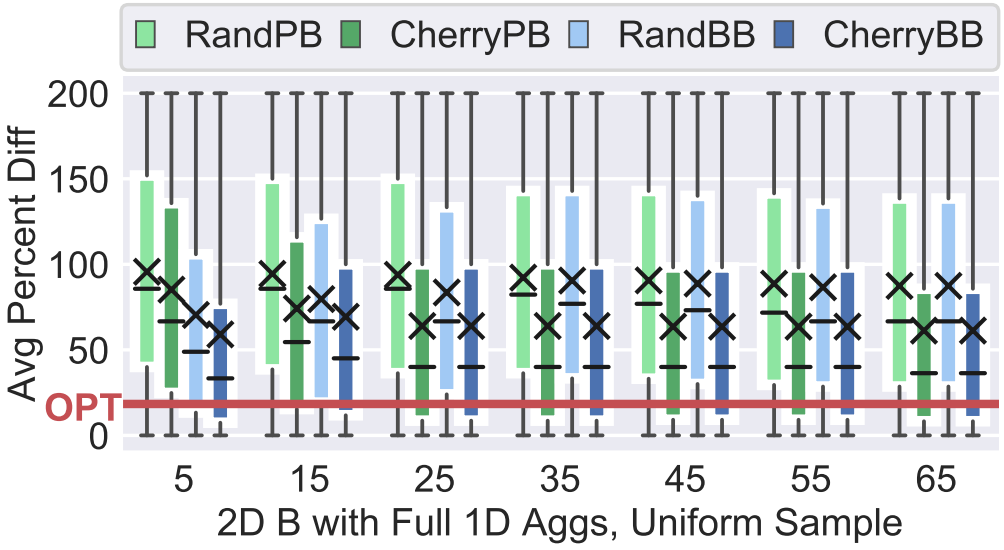


Figure 3.18: Average percent difference of 100 random point queries using a 10 percent uniform sample of the CHILD dataset with full 1D aggregates as more 2D aggregates are added using Prune and Rand.

Method	RW	SB	AB	SS	BS	BB
Runtime (10^{-3} s)	25.3	2.49	1.97	2.45	2.26	2.07

Table 3.8: Average query execution time of 100 random point queries over SR159 with 4 2D aggregates .

3.5.9 Execution Time

Lastly, we examine the query execution time and solving time of the different approaches. We run all timing experiments on the `IMDB` SR159 sample as `IMDB` has the larger active query domain (query times are approximately the same for `Flights` SCorners). For the query execution time, we run 100 random points queries.

Table 3.8 shoes the average query execution time of the six methods. As the query execution time does not noticeably change as we add more 2D aggregates, we only show results for 4 2D aggregates.

As the main bottle neck to using these methods is the solving time, we plot the time it takes to learn the structure for BB (the solving times of the other Bayesian methods are comparable or faster) and the time it takes to learn the parameters of LinReg, IPF, and BB in Figure 3.19 and Figure 3.20 as we increase the number of aggregates for `Flights` SCorners and `IMDB` SR159. The solving time is shown as a solid line while the structure learning time for BB is a dashed line.

The general trend is that LinReg is the fastest, followed closely by IPF, and then by BB. This is unsurprising as performing constrained optimization over our BN is more computationally complex than IPF, a simple multiplicative reweight procedure.

We further see that overall the times for parameter learning for SCorners is slower than for SR159. This is because the active domain of the attributes covered by the aggregates is larger for `Flights` than `IMDB` (see Table 3.4 and Table 3.5 for active domain information). A larger active domain means there are more aggregate values used to constrain the models and larger summations in our BN constrained optimization. Recall that BN parameters over nodes not in any aggregate are directly set from an aggregate query over the sample, which

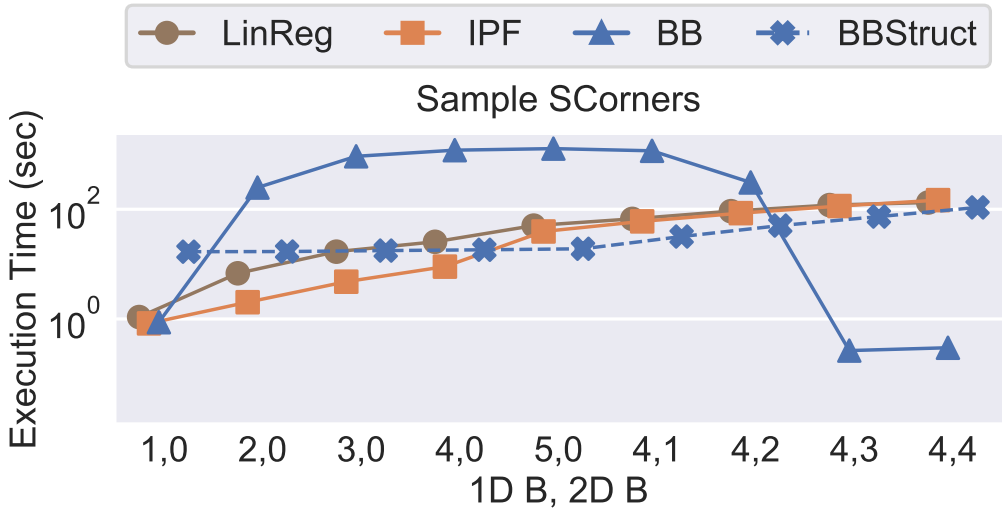


Figure 3.19: Log scale learning times in seconds for LinReg, IPF, and BB using SCorners sample as 1D and 2D aggregates are added.

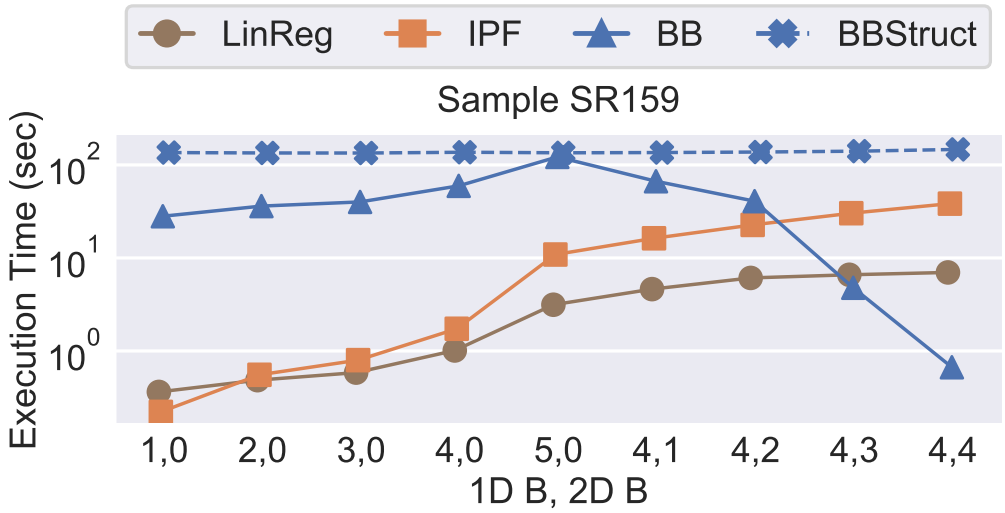


Figure 3.20: Log scale learning times in seconds for LinReg, IPF, and BB using SR159 sample as 1D and 2D aggregates are added.

is very efficient. Therefore, even if the active domain of some attributes are large, e.g., the active domain of `MT` and `N` in `IMDB`, as they are not included in any aggregates, they do not slow down solving time.

This is not true of structure learning. In structure learning, the most expensive operation is scoring a possible edge, which requires an aggregate, group by query over some set of

attributes in the aggregates or sample. As the aggregates over the attributes with the largest active domain is going to be the slowest computation, structure learning of IMDB is slower than for Flights. However, structure learning is independent of the number of aggregates and does not increase significantly as more aggregates are added because all edges are still considered in phase 2, the sample learning phase.

While the structure learning time for BB is constant across datasets, the solver time for BB varies drastically as we change aggregates. The solver time increases as we increase the number of 1D aggregates because we are adding more constraints to our model. Surprisingly, as we add more 2D aggregates, the solving time of BB decreases. This is because as we add more 2D constraints to our model, the constraint solver has more direct equality constraints which are instantaneous to solve for.

To show this trend more directly, Figure 3.21 and Figure 3.22 show the average percent difference for 100 random point queries compared to total solver time (structure learning plus parameter learning) for BB and IPF on SCorners and SR159 while using different combinations of 1D and 2D aggregates. Specifically, we use our pruning technique to add one to four 2D aggregates that cover from one up to five attributes[6].

For SCorners, we see that there is a cluster of BB methods that are as fast as IPF and can achieve low error. This corresponds to when the most 2D aggregates are added to the model. For SR159, IPF is always faster than BB to solve, but BB is capable of achieving lower error. This is because the IMDB sample is going to contain fewer distinct values of the population than the Flights sample. The samples are all ten percent samples but the active domain of the IMDB data is larger. Therefore, for random point queries, a tuple is less likely to be in the IMDB sample than Flights sample. Further, there is an optimal spot around 150 seconds of solving time for BB which achieves the lowest error. This is when the

[6]If we use a numeric index from 1 to 5 into the 5 covered attributes, we choose different aggregates that cover the following sets of attributes: {{1},{2},{3},{4},{5},{1,2},{4,5},{1,2,3},{3,4,5},{1,2,3,4},{2,3,4,5},{1,2,3,4,5}}. For each attribute set, we run our pruning technique to choose one, two, three, and four 2D aggregates, if applicable (we can't add 4 unique 2D aggregates for only two attributes). If any of the attributes are not covered by a 2D aggregates, we add 1D aggregates so all attributes are covered.

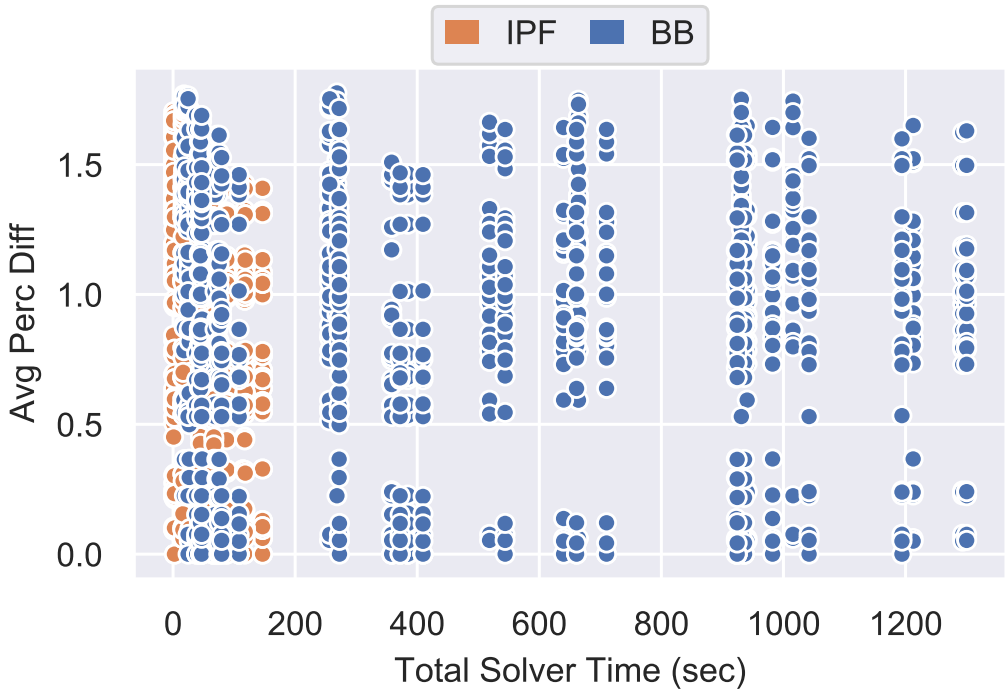


Figure 3.21: Average percent difference versus total solver time in seconds for IPF and BB on SCorners for various 1D and 2D aggregates.

most 2D aggregates are added to the model.

3.6 Discussion

As explained in Chapter 1, the Principle of Maximum Entropy is underlying both our sample reweighting and our Bayesian network technique. We address why this is true for each technique separately.

3.6.1 Maximum Entropy in IPF

The truly surprising connection is that the IPF algorithm is the same algorithm as the maximum entropy algorithm because IPF maximizes entropy [89, 20]. While we leave the detailed proof to [44], we are going to show how the IPF reweighted sample can be formulated as a MaxEnt problem and how the MaxEnt distribution can be solved for using IPF.

Recall from Section 2.1 that the distribution learned from maximizing entropy subject

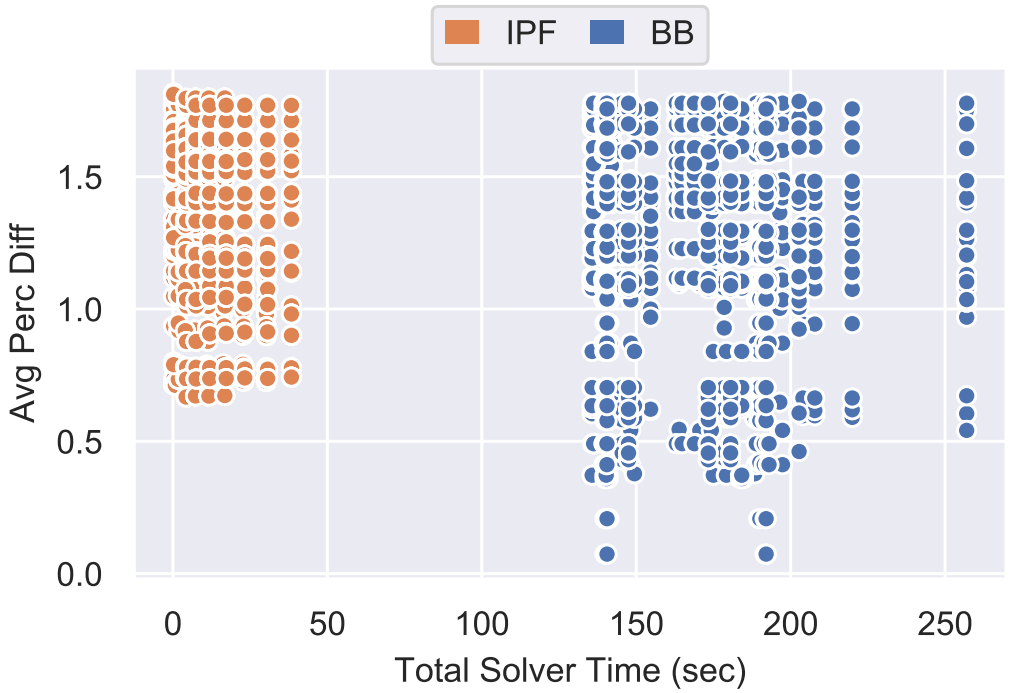


Figure 3.22: Average percent difference versus total solver time in seconds for IPF and BB on SR159 for various 1D and 2D aggregates.

to a set of expected value constraints over the set of possible worlds, PWD, takes the form shown in Equation 2.2, which is

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

where ϕ_j is a function on I, θ_j is a parameter to be learned, and Z is the normalization constant.

It turns out that if we have some known distribution q that we want our learned probability to be "close" to in terms of KL divergence, instead of maximizing the entropy, we can minimize the KL divergence to q subject to the same expected value constraints [13]. The solution takes the following form

$$\Pr(I) = \frac{q(I)}{Z} \exp\left(\sum_{j=1}^{k} \theta_j \phi_j(I)\right) \tag{3.3}$$

where Z is still the normalization constant equivalent to $\sum_I \Pr(I)$. The only difference between this and Equation 2.2 is the $q(I)$.

Equation 2.2 and Equation 3.3 together show that maximizing entropy is equivalent to minimizing the KL divergence to the uniform distribution where $q_{unif}(I) = \frac{1}{|PWD|}$.

We are going to define $q(I)$ is as follows. For each possible tuple t, assign a weight to t, denoted $w(t)$. $q(I)$ is then defined as

$$q(I) = \frac{\prod_{t \in I} w(t)}{\sum_{I \in PWD} \prod_{t \in I} w(t)}. \tag{3.4}$$

As an example, to get q_{unif}, we assign a weight of 1 to each possible tuple, i.e., $w_{unif}(t) = 1$.

Using this formulation, we can solve for the IPF weights by setting the weight of each possible tuple to the number of times it appears in our sample S, i.e., $w(t) = |\sigma_t(S)|$. Then we minimize the KL divergence with respect to the q defined in Equation 3.4 with the population aggregates as expected value constraints (like we do for EntropyDB) and get the MaxEnt solution. This MaxEnt solution gives us a probability distribution. The last step to set the final IPF weight of each tuple t in S to be $w_{final}(t) = \mathbb{E}[\langle \mathbf{t}, \mathbf{I} \rangle]$ where $\mathbf{t}$ is the associated linear query of tuple t.

The intuition behind why this works is that when we update the weights for a particular constraint in IPF, we are really updating a single parameter α in our MaxEnt model by the same multiplicative update. In IPF, a tuple's final weight is the product of its initial weight by all the weight updates it receives over the course of the algorithm. This is equivalent to its initial weight times to the product of the α parameters associate with it from the MaxEnt algorithm, i.e., $\mathbb{E}[\langle \mathbf{t}, \mathbf{I} \rangle]$.

We just showed how IPF can be solved with the MaxEnt framework. We will now show how the MaxEnt distribution parameters (the set of αs) can be solved with IPF. First, the same aggregate constraints are used as the expected value constraints. Second, we need to define our starting dataset. In this case, the initial dataset that starts IPF is a dataset of all possible tuples where each tuple has a weight of 1 (just like when defining q_{unif}). IPF

then proceeds as normal. The only difference being that instead of just updating the weights of each tuple, we keep track of the multiplicative updates associated with each aggregate constraint. This is equivalent to keeping track of the values of α. At the end, we get the final set of parameters to our MaxEnt model.

It's important to realize the major limitation to using IPF for the problem posed in Chapter 2. To build a probability distribution with just aggregate constraints using IPF, the starting IPF dataset is the dataset of all possible tuples. This is exponentially large because no tuples receive a weight of zero. They all have a weight of one. In traditional IPF starting with a sample, many tuples that could exist in the active domain have a weight of zero and do not exist. IPF cannot alter a weight of zero, meaning it can ignore those tuples. This fundamental issue makes solving tractable and efficient when there is an initial sample. Without an initial sample, we need to use more advanced optimization methods, like those presented in Chapter 2.

3.6.2 Maximum Entropy in Bayesian Networks

As explained in Section 2.7.4, there is a connection between the MaxEnt principle and maximum likelihood over graphical models. It can be shown that the MaxEnt distribution given expected value constraints is the dual of the maximum likelihood model of the exponential family given sufficient statistics [124]. Further, as graphical models are equivalent to certain exponential families [58], we get that the MaxEnt distribution given expected value constraints is the dual of the maximum likelihood given graphical model structural constraints [10]. Note, in this case, the duality gap is zero (meaning they are equivalent problems) as both MaxEnt and maximum likelihood are convex.

This implies, as further proven in [128], that Bayesian networks are really MaxEnt distributions given Bayesian network structural constraints.

3.7 Conclusion

We present THEMIS, the first prototype open world database system that uses a biased sample and population-level aggregate information to answer queries approximately as if asked over the population. More importantly, our data debiasing is automatic. THEMIS's hybrid approach merges sample reweighting with population probabilistic modeling to achieve a 70 percent improvement in the median error when compared to uniform reweighting for heavy hitter queries. Further, as shown in Figure 3.15, THEMIS is robust to differences in the support between the sample and population.

Chapter 4

RELATED WORK

4.1 Related Work on Approximate Query Processing Using a Probabilistic Model

Although there has been work in the theoretical aspects of probabilistic databases [116], 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 [94] 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 [25] 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 THEMIS also learns a Markov network, our features are count queries over instances while in [25], they only use boolean valued features.

Our work is also similar to that by Suciu and Ré [111] 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 [87, 37, 96]. One main AQP technique is to use precomputed samples [15, 87, 37, 14]. In the work by Chaudhiri

et al. [35], they precompute samples that minimize the errors due to variance in the data for a specific set of queries they predict. The work by [19] chooses multiple samples to use in query execution but only considers single column stratifications. VerdictDB [106] 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 [50] 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 [86] which generates a unified synopses from a set of samples to answer queries approximately within a predefined error bound. The later work of BlinkDB [16] 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 [78] 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 Themis 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 [34]. Much like Themis answers queries directly on a probability distribution, the system in [34] 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 [121] which builds a deep learning model in order to draw better samples for AQP. They first build a collection varia-

tional 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++ [107] 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 [57] 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 [21])[41, 87]. 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 [130] 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 THEMIS.

[105] and [80] both use deep learning networks to predict cardinalities for better query optimization. While THEMIS 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 [105] combines learning cardinalities with learning the best representation of a subquery.

4.2 Related Work on Automatic Debiasing

Our technique is primarily related to population synthesis. Population synthesis' goal is to directly generate a population dataset from a sample and either historical population data [90] or aggregate data (typically geographical aggregates) using techniques such as IPF, Bayesian networks, constrained optimization, or MCMC and Gibb's sampling [89, 23, 97, 117, 54, 82]. THEMIS, however, combines two different techniques, does not assume the sample is representative of the population, and more accurately answers queries over tuples not in the sample.

THEMIS is similar to bootstrapping, which is a resampling technique for understanding sample bias for approximate query processing [37, 96, 87], but THEMIS does not suffer from the large overhead of doing bootstrapping and does not assume the sample is representative of the population.

The papers from Van den Broeck and Martin Grohe [33, 62], extend probabilistic databases to assume an open world and allow unknown values to exist. While THEMIS also assumes an open world, we are not a probabilistic database.

From a data integration standpoint, THEMIS is closely related to answering queries over views (samples) [84, 64] where the views do not contain complete information. However, we attempt to model the data missing from the data sources whereas data integration deals with knowing when answers are certain or not.

In regards to data cleaning [74, 36, 126], while THEMIS is trying to infer missing values from the sample, we are missing entire rows, not just attribute values.

There has also been a lot of research in the machine learning community in two related areas: one class classification [79, 61] and learning from aggregate labels [101, 28, 38]. The main difference is that THEMIS is trying to learn a classifier with aggregate data and does not have aggregate labels of both classes, i.e., in the sample and not in the sample.

Our method, in essence, calculates a propensity score for a record [95, 112]. However, standard propensity score techniques require data that is not in the sample to be given,

which we do not have. A possible solution is to generate the data outside of the sample [66]. We leave this possible technique as future work.

The work of [40] is a particular subset of our problem. That work tries to take into account unknown unknown values when estimating aggregate queries. Our goal is similar, but we take a machine learning approach to re-weight samples rather than estimating missing values.

THEMIS is similar to [55, 135] which tries to remove bias from machine learning algorithms, i.e. make socially fair classifiers. Our research, however, does not have protected attributes nor does it have access to the entire population, but our methods could be used to help achieve fairness by, for example, forcing the aggregates over protected attributes to be uniform.

Considering selection bias and machine learning is the work of [71, 132]. THEMIS, however, only has access to the biased test set, i.e., sample, and does not have sample probabilities to use.

Although not concerned with debiasing data, the work of unioning tables [102] aims to find attributes among various tables that are unionable. The key similarity to THEMIS is that the tables are from different sources with different biases. Our research takes the output of [102], a set of unionable tables, and debiases them.

Also using aggregates as constraints is [123]. They use aggregates to constraint query answers; e.g., select tuples such that the average of the selection is below some value. The work in [93] discusses using Bayesian networks to approximate a data cube. Our work is similar to both of these except our goal is not to merely answer queries but to also debias a sample.

Similar to how THEMIS treats the biased samples as first class citizens, FactorBase [110] and BayesStore [125, 127] do the same with graphical models and probabilistic inference. While they both utilize Bayesian networks to model data, their goal is not data debiasing.

The work of [59] uses Bayesian networks over relations to do selectivity estimation for queries. THEMIS also uses Bayesian networks but is not concerned with multiple relations

or selectivity estimation.

[99] performs conjunctive query selectivity estimation using both samples and synopses. THEMIS also uses samples and synopses where our synopses are of the form of population histograms. Our overall goal, however, is to debias the data. This is different than selectivity estimation, and unlike selectivity estimation, we do not control how we sample.

Chapter 5

CONCLUSION AND FUTURE DIRECTIONS

In this modern age of easily accessible data samples and aggregate reports, data scientists are able to use these sources to ask questions over a population of data they do not have access to. While there are specialized, one-off techniques for merging samples and aggregates to model and approximately query the population, there is yet to be a general, automatic system for doing so. As traditional database management systems assumed a closed world, they are incapable of helping data scientists analyze an unknown population.

In this dissertation, we posit that an open world database system can provide the much needed support for data scientists by automating this population modeling. Data scientist simply insert a biased sample and population aggregates and then can issue queries over the population, receiving approximate answers.

We take the first step towards building an open world database system by answering two main research questions. The first is how to build a probabilistic model of the population using optimal aggregates. Framing the problem as one of building a probabilistic approximate query processing system, we develop and build a prototype system call EntropyDB. Using the Principle of Maximum Entropy and two-dimensional aggregate constraints, we solve for a probabilistic model and answer queries over this model directly. We show that EntropyDB performs as well as sampling on heavy hitter queries and can more accurately than sampling detect if a tuple exists or not. Further, EntropyDB maintains interactive query response times.

The second question we answer is how to automatically debias an arbitrary sample using population aggregates. We develop a system, Themis, that combines sample reweighting and probabilistic modeling to answer arbitrary queries over the population approximately

at interactive speeds. We demonstrate that THEMIS is more accurate at answering queries than uniform reweighting, iterative proportional fitting, and a variety of Bayesian network modeling techniques. Further, THEMIS is robust to samples that do not have the same support as the population; i.e., some tuples have no chance of being sampled. This robustness is lacking in the gold standard population synthesis technique of iterative proportional fitting.

The main, high-level open question that remains from this work is how to build a full-functioning, end-to-end open world database system. This system not only needs to be scalable and interactive but also needs to handle multiple different biased samples at once, to answer a large variety of queries beyond point queries, and to incorporate hierarchical aggregates.

To build a population model from multiple different samples, we will turn to the research in propensity score estimation for inspiration [95, 18]. Typically used to understand the differences in medical treatment groups, propensity score estimation aims to estimate the probability of an individual being included in a sample from comparing it to matching individuals in other samples. We will extend this idea to match tuples in one sample to tuples in other samples to learn the sampling probability. The challenges are how to extend this work to handle arbitrary samples where individuals may not match.

To answer a larger variety of queries, we will research and develop techniques to answer queries over probabilistic models that do not involve materializing a sample. While the work in [122, 119] discusses how to handle joins over probabilistic graphical models, it is still an open question how to handle top-k and generic group by queries.

Lastly, as hierarchical aggregates are common in demography research (e.g., aggregates over county and city and state), we need to extend our model to handle these nested aggregates. Iterative proportional fitting has been extended to hierarchical aggregates [98], and there has been work on hierarchical Bayesian networks [63]. However, it is still an open research question as how these two techniques can be merged into a unified framework.

Beyond building an open world database system, the other future directions for this work are to extend bias correction into bias detection. Can we help a user explain how their data

is bias? Further, can we use the debiasing work to improve fairness in datasets or help machine learning models learn about misrepresented or underrepresented subpopulations in a training set.

The overall takeaway from this dissertation is that building an open world database system is not only a relevant problem but a challenging one for the database community. Such a system will help automate the data analytics pipeline so that data scientists can more efficiently answer questions over their data.

BIBLIOGRAPHY

[1] https://www.congress.gov/115/bills/s760/BILLS-115s760is.pdf.

[2] http://infusecp.mimas.ac.uk.

[3] https://pdf.ic3.gov/2017_IC3Report.pdf.

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

[5] https://www.transtats.bts.gov/.

[6] http://www.bnlearn.com/bnrepository/discrete-medium.html.

[7] American community survey. https://www.census.gov/programs-surveys/acsv.

[8] Data.world. https://data.world.

[9] Large synoptic survey telescope. http://www.lsst.org/.

[10] Maximum entropy and maximum likelihood (slides). https://cedar.buffalo.edu/~srihari/CSE674/Chap20/20.4MNmaxentropy.pdf.

[11] These are the states with the longest and shortest commutes — how does yours stack up? https://www.cnbc.com/2018/02/22/study-states-with-the-longest-and-shortest-commutes.html.

[12] Twitter sample api. https://developer.twitter.com/en/products/tweets/sample.html.

[13] Ali Abbas, Andrea H Cadenbach, and Ehsan Salimi. A kullback–leibler view of maximum entropy and maximum log-probability methods. *Entropy*, 19(5):232, 2017.

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

[15] Swarup Acharya, Phillip B Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. The aqua approximate query answering system. In *ACM Sigmod Record*, volume 28, pages 574–576. ACM, 1999.

[16] Sameer 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.

[17] David A Applegate, Gruia Calinescu, David S Johnson, Howard Karloff, Katrina Ligett, and Jia 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.

[18] Peter C Austin. An introduction to propensity score methods for reducing the effects of confounding in observational studies. *Multivariate behavioral research*, 46(3):399–424, 2011.

[19] Brian Babcock, Surajit Chaudhuri, and Gautam 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.

[20] Hillel Bar-Gera, Karthik Konduri, Bhargava Sana, Xin Ye, and Ram M Pendyala. Estimating survey weights with multiple constraints using entropy optimization methods. In *88th Annual Meeting of the Transportation Research Board, Washington, DC*, 2009.

[21] Ziv Bar-Yossef, TS Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer, 2002.

[22] Jean-François Beaumont. A new approach to weighting and inference in sample surveys. *Biometrika*, 95(3):539–553, 2008.

[23] Richard J Beckman, Keith A Baggerly, and Michael D McKay. Creating synthetic baseline populations. *Transportation Research Part A: Policy and Practice*, 30(6):415–429, 1996.

[24] Michael Behrisch, Benjamin Bach, Nathalie Henry Riche, Tobias Schreck, and Jean-Daniel Fekete. Matrix reordering methods for table and network visualization. In *Computer Graphics Forum*, volume 35, pages 693–716. Wiley Online Library, 2016.

[25] Jessa Bekker, Jesse Davis, Arthur Choi, Adnan Darwiche, and Guy Van den Broeck. Tractable learning for complex probability queries. In *Advances in Neural Information Processing Systems*, pages 2242–2250, 2015.

[26] Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. A maximum entropy approach to natural language processing. *Comput. Linguist.*, 22(1), 1996.

[27] Jelke Bethlehem et al. The rise of survey sampling. 2009.

[28] Avradeep Bhowmik, Joydeep Ghosh, and Oluwasanmi Koyejo. Generalized linear models for aggregated data. In *Artificial Intelligence and Statistics*, pages 93–101, 2015.

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

[30] Brian Buck and Vincent A Macaulay. *Maximum entropy in action: a collection of expository essays*. Oxford University Press, USA, 1991.

[31] József Bukszár and Andras Prekopa. Probability bounds with cherry trees. *Mathematics of Operations Research*, 26(1):174–192, 2001.

[32] Luis M de Campos. A scoring function for learning bayesian networks based on mutual information and conditional independence tests. *Journal of Machine Learning Research*, 7(Oct):2149–2187, 2006.

[33] Ismail Ilkan Ceylan, Adnan Darwiche, and Guy Van den Broeck. Open-world probabilistic databases. In *Description Logics*, 2016.

[34] Kaushik Chakrabarti, Minos Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Approximate query processing using wavelets. *The VLDB Journal—The International Journal on Very Large Data Bases*, 10(2-3):199–223, 2001.

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

[36] Surajit Chaudhuri, Anish Das Sarma, Venkatesh Ganti, and Raghav Kaushik. Leveraging aggregate constraints for deduplication. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 437–448. ACM, 2007.

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

[38] Bee-Chung Chen, Lei Chen, Raghu Ramakrishnan, and David R Musicant. Learning from aggregate views. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 3–3. IEEE, 2006.

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

[40] Yeounoh Chung, Michael Lind Mortensen, Carsten Binnig, and Tim Kraska. Estimating the impact of unknown unknowns on aggregate query results. *ACM Transactions on Database Systems (TODS)*, 43(1):3, 2018.

[41] Graham Cormode, Minos Garofalakis, Peter J Haas, Chris Jermaine, et al. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases*, 4(1–3):1–294, 2011.

[42] Corinna Cortes, Mehryar Mohri, Michael Riley, and Afshin Rostamizadeh. Sample selection bias correction theory. In *International Conference on Algorithmic Learning Theory*, pages 38–53. Springer, 2008.

[43] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim 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.

[44] Imre Csiszár. I-divergence geometry of probability distributions and minimization problems. *The Annals of Probability*, pages 146–158, 1975.

[45] Nilesh Dalvi, Christopher Ré, and Dan Suciu. Probabilistic databases: diamonds in the dirt. *Communications of the ACM*, 52(7):86–94, 2009.

[46] Cassio P De Campos, Yan Tong, and Qiang Ji. Constrained maximum likelihood learning of bayesian networks for facial action recognition. In *European Conference on Computer Vision*, pages 168–181. Springer, 2008.

[47] W Edwards Deming and Frederick F Stephan. On a least squares adjustment of a sampled frequency table when the expected marginal totals are known. *The Annals of Mathematical Statistics*, 11(4):427–444, 1940.

[48] Amol Deshpande, Minos N. Garofalakis, and Rajeev Rastogi. Independence is good: Dependency-based histogram synopses for high-dimensional data. In *SIGMOD Conference*, 2001.

[49] Amol Deshpande and Samuel Madden. Mauvedb: supporting model-based user views in database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 73–84. ACM, 2006.

[50] Bolin Ding, Silu Huang, Surajit Chaudhuri, Kaushik Chakrabarti, and Chi Wang. Sample+seek: Approximating aggregates with distribution precision guarantee. In *Proc. SIGMOD*, pages 679–694, 2016.

[51] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[52] Cynthia Dwork. Differential privacy and the us census. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 1–1. ACM, 2019.

[53] Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toni Pitassi, Omer Reingold, and Aaron Roth. Generalization in adaptive data analysis and holdout reuse. In *Advances in Neural Information Processing Systems*, pages 2350–2358, 2015.

[54] Bilal Farooq, Michel Bierlaire, Ricardo Hurtubia, and Gunnar Flötteröd. Simulation based population synthesis. *Transportation Research Part B: Methodological*, 58:243–263, 2013.

[55] Michael Feldman, Sorelle A Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 259–268. ACM, 2015.

[56] Nir Friedman, Iftach Nachman, and Dana Peér. Learning bayesian network structure from massive datasets: the "sparse candidate" algorithm. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 206–215. Morgan Kaufmann Publishers Inc., 1999.

[57] Alex Galakatos, Andrew Crotty, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. Revisiting reuse for approximate query processing. *Proceedings of the VLDB Endowment*, 10(10):1142–1153, 2017.

[58] Dan Geiger and Christopher Meek. Graphical models and exponential families. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 156–165. Morgan Kaufmann Publishers Inc., 1998.

[59] Lise Getoor, Benjamin Taskar, and Daphne Koller. Selectivity estimation using probabilistic models. In *ACM SIGMOD Record*, volume 30, pages 461–472. ACM, 2001.

[60] Phillip B Gibbons, Viswanath Poosala, Swarup Acharya, Yair Bartal, Yossi Matias, S Muthukrishnan, Sridhar Ramaswamy, and Torsten Suel. Aqua: System and techniques for approximate query answering. *Bell Labs TR*, 1998.

[61] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[62] Martin Grohe and Peter Lindner. Probabilistic databases with an infinite open-world assumption. *arXiv preprint arXiv:1807.00607*, 2018.

[63] Elias Gyftodimos and Peter A Flach. Hierarchical bayesian networks: an approach to classification and learning for structured data. In *Hellenic Conference on Artificial Intelligence*, pages 291–300. Springer, 2004.

[64] Alon Y Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[65] Moritz Hardt and Guy 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.

[66] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge & Data Engineering*, (9):1263–1284, 2008.

[67] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. Online aggregation. In *ACM SIGMOD Record*, volume 26, pages 171–182. ACM, 1997.

[68] Max Henrion. Propagating uncertainty in bayesian networks by probabilistic logic sampling. In *Machine Intelligence and Pattern Recognition*, volume 5, pages 149–163. Elsevier, 1988.

[69] Gzyl Henryk. *The method of maximum entropy*, volume 29. World scientific, 1995.

[70] Anup Hosangadi, Farzan Fallah, and Ryan Kastner. Factoring and eliminating common subexpressions in polynomial expressions. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, 2004.

[71] Jiayuan Huang, Arthur Gretton, Karsten Borgwardt, Bernhard Schölkopf, and Alex J Smola. Correcting sample selection bias by unlabeled data. In *Advances in neural information processing systems*, pages 601–608, 2007.

[72] Zengyi Huang and Paul Williamson. A comparison of synthetic reconstruction and combinatorial optimisation approaches to the creation of small-area microdata. *Department of Geography, University of Liverpool*, 2001.

[73] Martin Idel. A review of matrix scaling and sinkhorn's normal form for matrices and positive maps. *arXiv preprint arXiv:1609.06349*, 2016.

[74] Ihab F Ilyas, Xu Chu, et al. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends® in Databases*, 5(4):281–393, 2015.

[75] Chris Jermaine, Subramanian Arumugam, Abhijit Pol, and Alin Dobra. Scalable approximate query processing with the dbo engine. *ACM Transactions on Database Systems (TODS)*, 33(4):23, 2008.

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

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

[78] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin 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.

[79] Shehroz S Khan and Michael G Madden. A survey of recent trends in one class classification. In *Irish conference on artificial intelligence and cognitive science*, pages 188–197. Springer, 2009.

[80] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677*, 2018.

[81] Kevin B Korb and Ann E Nicholson. *Bayesian artificial intelligence*. CRC press, 2010.

[82] Der-Horng Lee and Yingfei Fu. Cross-entropy optimization model for population synthesis in activity-based microsimulation models. *Transportation Research Record*, 2255(1):20–27, 2011.

[83] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proceedings of the VLDB Endowment*, 9(3):204–215, 2015.

[84] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM, 2002.

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

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

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

[88] Sharon L Lohr. *Sampling: Design and Analysis: Design and Analysis*. Chapman and Hall/CRC, 2019.

[89] Robin Lovelace, Mark Birkin, Dimitris Ballas, and Eveline van Leeuwen. Evaluating the performance of iterative proportional fitting for spatial microsimulation: new tests for an established technique. *Journal of Artificial Societies and Social Simulation*, 18(2), 2015.

[90] Michael C Lovell. Seasonal adjustment of economic time series and multiple regression analysis. *Journal of the American Statistical Association*, 58(304):993–1010, 1963.

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

[92] Dimitris Margaritis. Learning bayesian network model structure from data. Technical report, Carnegie-Mellon Univ Pittsburgh Pa School of Computer Science, 2003.

[93] Dimitris Margaritis, Christos Faloutsos, and Sebastian Thrun. Netcube: A scalable tool for fast data mining and compression. In *VLDB*, pages 311–320, 2001.

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

[95] Daniel F McCaffrey, Beth Ann Griffin, Daniel Almirall, Mary Ellen Slaughter, Rajeev Ramchand, and Lane F Burgette. A tutorial on propensity score estimation for multiple treatments using generalized boosted models. *Statistics in medicine*, 32(19):3388–3414, 2013.

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

[97] Kirill Müller. *A generalized approach to population synthesis*. PhD thesis, ETH Zurich, 2017.

[98] Kirill Müller and Kay W Axhausen. Multi-level fitting algorithms for population synthesis. *Arbeitsberichte Verkehrs-und Raumplanung*, 821, 2012.

[99] Magnus Müller, Guido Moerkotte, and Oliver Kolb. Improved selectivity estimation by combining knowledge from sampling and synopses. *Proceedings of the VLDB Endowment*, 11(9):1016–1028, 2018.

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

[101] David R Musicant, Janara M Christensen, and Jamie F Olson. Supervised learning by training on aggregate outputs. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 252–261. IEEE, 2007.

[102] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. Table union search on open data. In *VLDB*, 2018.

[103] Radu Stefan Niculescu. Exploiting parameter domain knowledge for learning in bayesian networks. *Technical Report CMU-TR-05-147*, 2005.

[104] Laurel Orr, Magdalena Balazinska, and Dan Suciu. Probabilistic database summarization for interactive data exploration. *Proceedings of the VLDB Endowment*, 10(10):1154–1165, 2017.

[105] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S Sathiya Keerthi. Learning state representations for query optimization with deep reinforcement learning. *arXiv preprint arXiv:1803.08604*, 2018.

[106] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. Verdictdb: universalizing approximate query processing. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1461–1476. ACM, 2018.

[107] Jinglin Peng, Dongxiang Zhang, Jiannan Wang, and Jian 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.

[108] Steven J Phillips, Miroslav Dudík, and Robert E Schapire. A maximum entropy approach to species distribution modeling. In *Proceedings of the twenty-first international conference on Machine learning*, page 83. ACM, 2004.

[109] Brian Proulx and Junshan Zhang. Modeling social network relationships via t-cherry junction trees. In *INFOCOM, 2014 Proceedings IEEE*, pages 2229–2237. IEEE, 2014.

[110] Zhensong Qian and Oliver Schulte. Factorbase: Multi-relational model learning with sql all the way. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–10. IEEE, 2015.

[111] Christopher Ré and Dan Suciu. Understanding cardinality estimation using entropy maximization. *ACM TODS*, 37(1):6, 2012.

[112] Paul R Rosenbaum and Donald B Rubin. Reducing bias in observational studies using subclassification on the propensity score. *Journal of the American statistical Association*, 79(387):516–524, 1984.

[113] Justin Ryan, Hanna Maoh, and Pavlos Kanaroglou. Population synthesis: Comparing the major techniques using a small, complete population of firms. *Geographical Analysis*, 41(2):181–203, 2009.

[114] Richard Sinkhorn. Diagonal equivalence to matrices with prescribed row and column sums. *The American Mathematical Monthly*, 74(4):402–405, 1967.

[115] Daniel Solow. Linear and nonlinear programming. *Wiley Encyclopedia of Computer Science and Engineering*, 2007.

[116] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. Probabilistic databases. *Synthesis Lectures on Data Management*, 3(2):1–180, 2011.

[117] Lijun Sun and Alexander Erath. A bayesian network approach for population synthesis. *Transportation Research Part C: Emerging Technologies*, 61:49–62, 2015.

[118] Tamás Szántai and Edith Kovács. Discovering a junction tree behind a markov network by a greedy algorithm. *Optimization and Engineering*, 14(4):503–518, 2013.

[119] Ben Taskar, Pieter Abbeel, and Daphne Koller. Discriminative probabilistic models for relational data. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 485–492. Morgan Kaufmann Publishers Inc., 2002.

[120] Yee Whye Teh and Max Welling. On improving the efficiency of the iterative proportional fitting procedure. In *AIStats*, 2003.

[121] Saravanan Thirumuruganathan, Shohedul Hasan, Nick Koudas, and Gautam Das. Approximate query processing using deep generative models. *arXiv preprint arXiv:1903.10000*, 2019.

[122] Kostas Tzoumas, Amol Deshpande, and Christian S Jensen. Efficiently adapting graphical models for selectivity estimation. *The VLDB Journal*, 22(1):3–27, 2013.

[123] Manasi Vartak, Venkatesh Raghavan, Elke A Rundensteiner, and Samuel Madden. Refinement driven processing of aggregation constrained queries. In *EDBT*, pages 101–112, 2016.

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

[125] Daisy Zhe Wang, Eirinaios Michelakis, Minos Garofalakis, and Joseph M Hellerstein. Bayesstore: managing large, uncertain data repositories with probabilistic graphical models. *Proceedings of the VLDB Endowment*, 1(1):340–351, 2008.

[126] Jiannan Wang, Sanjay Krishnan, Michael J Franklin, Ken Goldberg, Tim Kraska, and Tova Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 469–480. ACM, 2014.

[127] Zhe Wang. *Extracting and Querying Probabilistic Information in BayesStore*. PhD thesis, UC Berkeley, 2011.

[128] Jon Williamson. Foundations for bayesian networks. In *Foundations of Bayesianism*, pages 75–115. Springer, 2001.

[129] Raymond Chi-Wing Wong and Ada Wai-Chee Fu. Mining top-k frequent itemsets from data streams. *Data Mining and Knowledge Discovery*, 13(2):193–217, 2006.

[130] Mingxi Wu and Christopher 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.

[131] Eunho Yang, Pradeep Ravikumar, Genevera I Allen, and Zhandong Liu. Graphical models via univariate exponential family distributions. *The Journal of Machine Learning Research*, 16(1):3813–3847, 2015.

[132] Bianca Zadrozny. Learning and evaluating classifiers under sample selection bias. In *Proceedings of the twenty-first international conference on Machine learning*, page 114. ACM, 2004.

[133] Emilio Zagheni, Venkata Rama Kiran Garimella, Ingmar Weber, et al. Inferring international and internal migration patterns from twitter data. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 439–444. ACM, 2014.

[134] Emilio Zagheni and Ingmar Weber. Demographic research with non-representative internet data. *International Journal of Manpower*, 36(1):13–25, 2015.

[135] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In *International Conference on Machine Learning*, pages 325–333, 2013.

Appendix A

T-CHERRY JUNCTION TREE

Algorithm 10 gives the detailed pseudo code of our modified t-cherry junction tree building algorithm. Given a dimension k to use as our max cluster size, budget of B aggregates to keep in Γ, attribute set $\mathcal{A} = \{A_1, \ldots, A_m\}$, and Γ, we output a set of clusters $\mathcal{C}'$. The attributes associated with each cluster are the attributes of the aggregates to keep in Γ. Therefore, we filter Γ so that each $\boldsymbol{\gamma}_i$ must be equal to the set of attributes associated with a cluster in $\mathcal{C}'$.

Algorithm 9 GENCLUSTERSEPARATORPAIRS

$\mathcal{C} = \varnothing$

for $c \in$ COMBINATIONS($\mathcal{A}$, k) **where** $attrs(c) \in \Gamma$ **do**

  for $v \in c$ **do**

    $S = c - v$ ▷ Separator

    $w = I(\Gamma_c) - I(\Gamma_S)$ ▷ Weight

    $P =$ `None` ▷ Parent

    $\mathcal{C} = \mathcal{C} \cup (c, S, P, v, w)$

return $\mathcal{C}$

Algorithm 10 Modified t-cherry tree.

$\mathcal{C}' = \varnothing$
$\mathcal{C} \leftarrow$ sorted(GenClusterSeparatorPairs)
$c \leftarrow \mathcal{C}[0]$
$\mathcal{A}' \leftarrow attrs(\mathcal{C}[0])$
$\texttt{idx} \leftarrow 1$
$i \leftarrow 0$ ▷ Number clusters seen since update
$t \leftarrow \texttt{False}$ ▷ New tree
while $|c| < B$ **do**
  $d \leftarrow \mathcal{C}[\texttt{idx}]$
  if $d.v \notin \mathcal{A}'$ **then**
    $P \leftarrow$ FindParent(d, c) ▷ Find P containing $d.S$, d's separator
    if $P \neq \texttt{None}$ **or** t **then**
      $d.P \leftarrow P$
      $\mathcal{A}' \leftarrow \mathcal{A}' \cup attrs(d)$
      $c \leftarrow c \cup d$
      $i \leftarrow 0$
      $t \leftarrow \texttt{False}$
  $\texttt{idx} \leftarrow (\texttt{idx} + 1)\%|\mathcal{C}|$
  $i \leftarrow i + 1$
  if $i = |\mathcal{C}|$ **then** ▷ Check if need to make new tree
    $t \leftarrow \texttt{True}$
    $\texttt{idx} \leftarrow 0$
  if $\mathcal{A}' \leftarrow \mathcal{A}$ **then** ▷ Check if need to reset attributes
    $\mathcal{C}' \leftarrow \mathcal{C}' \cup c$
    $\mathcal{C} \leftarrow$ PruneClusters($\mathcal{C}, c$) ▷ Prune clusters to have no duplicates
    $\mathcal{A} \leftarrow attrs(\mathcal{C})$
    $\mathcal{A}' \leftarrow \varnothing$
    $c \leftarrow \varnothing$
    $t \leftarrow \texttt{True}$
    $\texttt{idx} \leftarrow 0$
  $\mathcal{C}' \leftarrow \mathcal{C}' \cup \mathcal{C}'$
end while
return $\mathcal{C}'$