THE UNIVERSITY OF CHICAGO

FAST AND RELIABLE MISSING DATA CONTINGENCY ANALYSIS WITH PREDICATE-CONSTRAINTS

A DISSERTATION SUBMITTED TO THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCE IN CANDIDACY FOR THE DEGREE OF MASTER

DEPARTMENT OF COMPUTER SCIENCE

BY

XI LIANG

CHICAGO, ILLINOIS
NOVEMBER 26, 2019
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>vii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>viii</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2 BACKGROUND</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Applications</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Scope and Implications</td>
<td>6</td>
</tr>
<tr>
<td>3 APPROACH</td>
<td>8</td>
</tr>
<tr>
<td>3.1 Predicate-Constraints</td>
<td>8</td>
</tr>
<tr>
<td>3.1.1 Predicate-Constraint</td>
<td>8</td>
</tr>
<tr>
<td>3.1.2 Predicate-Constraint Sets</td>
<td>10</td>
</tr>
<tr>
<td>3.1.3 Discussion</td>
<td>11</td>
</tr>
<tr>
<td>4 METHODOLOGY</td>
<td>12</td>
</tr>
<tr>
<td>4.1 Calculating</td>
<td>12</td>
</tr>
<tr>
<td>4.1.1 Cell Decomposition</td>
<td>13</td>
</tr>
<tr>
<td>4.1.2 Integer-Linear Program</td>
<td>15</td>
</tr>
<tr>
<td>4.1.3 Complexity</td>
<td>17</td>
</tr>
<tr>
<td>4.1.4 Special Cases and Discussion</td>
<td>18</td>
</tr>
<tr>
<td>5 EXPERIMENTS AND EVALUATIONS</td>
<td>19</td>
</tr>
<tr>
<td>5.1 Experimental Setup</td>
<td>19</td>
</tr>
<tr>
<td>5.1.1 Sampling</td>
<td>20</td>
</tr>
<tr>
<td>5.1.2 Generative Model</td>
<td>20</td>
</tr>
<tr>
<td>5.1.3 Equiwidth Histogram</td>
<td>21</td>
</tr>
<tr>
<td>5.1.4 PCs</td>
<td>21</td>
</tr>
<tr>
<td>5.1.5 Datasets</td>
<td>21</td>
</tr>
<tr>
<td>5.2 Probabilistic Confidence Intervals are Often Unreliable on Real Data</td>
<td>24</td>
</tr>
<tr>
<td>5.3 PCs are Competitively Accurate</td>
<td>25</td>
</tr>
<tr>
<td>5.3.1 Intel Wireless Dataset</td>
<td>25</td>
</tr>
<tr>
<td>5.3.2 Airbnb at New York City Dataset</td>
<td>26</td>
</tr>
<tr>
<td>5.3.3 Border Crossing Dataset</td>
<td>26</td>
</tr>
<tr>
<td>5.3.4 PCs are Data Efficient</td>
<td>27</td>
</tr>
<tr>
<td>5.4 Worst-Case Optimal Estimates On Joins</td>
<td>27</td>
</tr>
<tr>
<td>5.4.1 Counting Triangles</td>
<td>28</td>
</tr>
<tr>
<td>5.4.2 Acyclic Joins</td>
<td>29</td>
</tr>
</tbody>
</table>
5.5 Optimizations for Overlapping PCs ........................................... 29
5.6 Non-Overlapping PCs Scale Effectively .................................... 31
5.7 PCs Are Relatively Robust To Noise ......................................... 32
5.8 PCs Are Not Sensitive to Query Selectivity ............................... 33
5.9 Handling MIN, MAX, and AVG Queries ................................. 33

6 DISCUSSION AND RELATED WORK ........................................... 36
   6.1 Related Work ............................................................... 36

7 SUMMARY ................................................................. 39

REFERENCES ................................................................. 40
LIST OF FIGURES

4.1 Predicates in a predicate-constraint set are possibly overlapping. The first step of the algorithm is to decompose a set of predicate-constraints into disjoint cells. 13

5.1 Over-estimation rate on random 100 COUNT(*) and SUM queries on the Intel Wireless dataset. Despite having a hard range guarantee, PCs are competitively accurate with statistical baselines. 26

5.2 Baseline performance on 100 COUNT(*) and SUM queries with predicate attributes on Latitude using the Airbnb NYC dataset. 27

5.3 Baseline performance on 100 COUNT(*) and SUM queries with predicate attributes on Port and Date using the Border crossing dataset. 28

5.4 Performance of CLT baselines and Chernoff baselines vs. different sample size. 29

5.5 We compare the bound derived by our approach (Corr-PC) with state of the art baseline Elastic Sensitivity on the triangle counting problem of different table sizes (TOP) and an acyclic join (BOTTOM). 30

5.6 Our optimizations reduce the number of cells evaluated during the cell decomposition phase by over a 1000x. 31

5.7 The run time needed to solve one query using partition of different size. 32

5.8 We investigate the sensitivity of Corr-PC, Overlapping-PC, and US-10n to different levels of noise. The failure rate of all approaches increases as we increase the noise level, but the PC baselines especially Overlapping-PC are more tolerable to the same level of noise. 33

5.9 Bound for queries with different selectivity. 34

5.10 With PC, an optimal bound can be derived for MIN and MAX queries. PC also generates competitive result for AVG Queries. 35
5.1 Over a 100 randomly chosen predicates, we record the failure rate in percentage of different “error bound” frameworks. A failure event is one where an observed outcome is outside the range of the returned bounds. For a 99% confidence interval, as used in our experiments, one would expect a 1% failure rate for the sampling-based frameworks—but is significantly higher in practice for small skewed datasets. PCs, and as a special case Histograms, are guaranteed not to fail if the assumptions are satisfied.
ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Prof. Sanjay Krishnan for his support and guidance on my research without which this thesis would not have been possible.

I also thank Prof. Michael Franklin and Prof. Aaron Elmore for their great support and invaluable constructive advice.

My sincere thanks also goes to Dr. Zechao Shang for his effort on this research project which – among other things – helped me addressing technical challenges and avoiding road blocks.

Finally, I want to thank my fellow labmates in the Database group for the stimulating discussions, valuable suggestions and encouraging atmosphere.
ABSTRACT

Today, data analysts largely rely on intuition to determine whether missing or withheld rows of a dataset significantly affect their analyses. We propose a framework that can produce automatic contingency analysis, i.e., the range of values an aggregate SQL query could take, under formal constraints describing the variation and frequency of missing data tuples. We describe how to process \texttt{SUM}, \texttt{COUNT}, \texttt{AVG}, \texttt{MIN}, and \texttt{MAX} queries in these conditions resulting in hard error bounds with testable constraints. We propose an optimization algorithm based on an integer program that reconciles a set of such constraints, even if they are overlapping, conflicting, or unsatisfiable, into such bounds. We also present a novel formulation of the Fractional Edge Cover problem to account for cases where constraints span multiple tables. Our experiments on 4 datasets against several statistical imputation and inference baselines show that statistical techniques can have a deceptively high error rate that is often unpredictable. In contrast, our framework offers hard bounds that are guaranteed to hold if the constraints are not violated. In spite of these hard bounds, we show competitive accuracy to statistical baselines.
CHAPTER 1
INTRODUCTION

The data stored in a database may differ from real-world truth in terms of both completeness and content. Such issues can arise due to data entry errors, inexact data integration, or software bugs [7]. As real-world data are rarely perfectly clean or complete, data scientists have to reason how potential sources of error may affect their analyses. Communicating these error modes and quantifying the uncertainty they introduce into a particular analysis is arguably as important as timely execution [16].

For example, suppose a data analyst has collected data from a temperature sensor over the span of several days. She is interested in computing the number of times that the sensor exceeded a temperature threshold. The data are stored in 10 partitions; one of which failed to load into the database due to parsing errors. The analyst can still run her query on the 9 available partitions, however, she needs to determine whether the loss of that partition may affect her conclusions.

Today, analysts largely rely on intuition to reason about such scenarios. The analyst in our example needs to make a judgment about whether the lost partition correlates with the attributes of interest, such as temperature, in any way. Such intuitive judgments, while commonplace, are highly problematic because they are based on assumptions that are often not formally encoded in any code or documentation. Simply reporting an extrapolated result does not convey any measure of confidence in how (in)accurate the result might be, and could hide the fact that some of the data were not used.

This paper defines a formal framework for specifying assumptions in a logical constraint language to describe what an analyst believes is true about withheld or missing rows of data. Given these constraints, we develop an algorithm for computing a range of values an aggregate query can take under those constraints (hereafter called a 1).

---

1. We use this term to differentiate a deterministic range with probabilistic confidence intervals.
which we call the Predicate-Constraint (PC) framework, facilitates several desirable outcomes: (1) the constraints are efficiently testable on historical data to determine whether or not they held true in the past, (2) the is calculated deterministically and guaranteed to bound the results if the constraints hold true in the future, (3) the framework can reconcile interacting, overlapping, or conflicting PCs by enforcing the most restrictive ones, and (4) the framework makes no distributional assumptions about past data resembling future data other than what is specified in the PCs. With this framework, a data scientist can produce a contingency analysis, i.e., the range of values the aggregate could take, under formally described assumptions about the nature of the unseen data. PCs can be checked, versioned, and tested just like any other analysis code—ultimately facilitating a more reproducible analysis methodology. While we focus on analyzing tables with missing rows, the framework can be trivially extended to model data corruption. We omit that discussion for the sake of brevity.

Predicate-Constraints are logical statements that constrain the range of values that a set of rows can take and the number of such rows within a predicate. We show that deriving the for a single “closed” predicate-constraint set can be posed as a mixed-integer linear program (MIP). If the query is unboundable given the constraints, the algorithm returns a failure. We show links to the Fractional Edge Cover bounds employed in the analysis of worst-case optimal joins when we consider constraints over multiple tables and contribute a new variant of the same problem which can be used to bound predicate-constraints [21]. The solver itself contains a number of novel optimizations, which we contribute such as early pruning of unsatisfiable search paths.

To the best of our knowledge, a direct competitor framework does not exist. While there is a rich history of what-if analysis [10] and how-to analysis [19], which characterize a database’s behavior under hypothetical updates, analyzing the effects of constrained missing rows on aggregate queries has been not been extensively studied. The closest such framework is the m-table framework [30], which has a similar expressiveness but no algorithm for computing
aggregate. Likewise, some of the work in data privacy solves a simplified version of the problem where there are no overlapping constraints or join conditions [32].

As such, we compare our framework to baselines that are rooted in statistical inference. Our baselines are given some amount of information about the distribution of the missing values either through data samples (example rows that are missing), attribute histograms (coarse distribution of attribute values), or probabilistic parameters. The comparison to statistical competitors shows that these guaranteed bounds provided by the PC framework are competitive with statistical estimates without the pitfalls of probabilistic confidence intervals and sensitivity to data skew. We also illustrate cases when the deterministic bounds computed by our framework are even more accurate than probabilistic ones.

In summary, we contribute:

1. A formal framework for contingency analysis over missing or withheld rows of data, where users specify constraints about the frequency and variation of the missing rows.

2. An optimization algorithm that reconciles a set of such constraints, even if they are overlapping, conflicting, or unsatisfiable, into a range of possible values that SUM, COUNT, AVG, MIN, and MAX SQL queries can take.

3. A novel formulation of the Fractional Edge Cover problem to account for cases where constraints span multiple tables and improve accuracy for natural joins.

4. Meta-optimizations that improve accuracy and/or optimization performance such as pruning unsatisfiable constraint paths.
CHAPTER 2
BACKGROUND

In this section, we motivate the general problem and describe a handful of application scenarios. We have a database that is missing some number of rows from each of its tables. The user is allowed to constrain the cardinality and values of those missing rows through a formal language of constraints, called Predicate Constraints. The framework determines for a given aggregate query the highest and lowest values the query result could take under the conditions stipulated by the constraints.

2.1 Applications

Application 1. Approximate Querying of Partitioned Data

Let’s consider an example similar to that in the introduction. In general, taking a truly uniform sample of data requires iterating through an entire dataset. This may not be feasible over very large or geo-distributed datasets. It may be far more efficient (but potentially erroneous) to simply consider the first $k$ partitions of a dataset and ignoring the remaining $N - k$. If these partitions are not random or are correlated with important attributes, the results could be very unreliable. Fortunately, many partitioned file formats provide metadata about what is contained in the partitioned such as the min and max attribute values, the number of rows, and the partitioning criteria. Normally, this metadata is used for data skipping [29] but it is exactly in constraint format expected by our framework—a predicate, a range, and a size. Thus, we can assess the impact of unprocessed partitions on an aggregate query result.
**Application 2. Non-Response Bias**

Non-response bias, or bias in those people who chose to respond to an opinion poll request, is considered one of the likely culprits of why election polls were so erroneous in the lead up to the 2016 U.S. election [27]. Accounting for different types of non-response bias is an ideal application of our framework. Let’s assume that we are given a table:

\[
\text{Vote}(\text{cand}, \text{age}, \text{race}, \text{income})
\]

where \text{cand} is a candidate, \text{age} is a numerical value describing the participant’s age, \text{race} describes the participant’s racial group, and \text{income} describes a participant’s income level. To calculate the result of the poll is simply an SQL query:

\[
\begin{align*}
\text{SELECT } & \text{COUNT}(1) \\
\text{FROM } & \text{Vote} \\
\text{GROUP BY } & \text{cand}
\end{align*}
\]

A polling agency may send out 100 polling requests to randomly chosen participants, only 80 of which are completely answered.

We would like to reason about how those missing 20 requests might change our polling results. PCs allow a data scientist to hypothesize different scenarios and observe how they affect the overall count. From historical experience, the data scientist might guess that: between 10–20\% of the rows are from the age 18–25 age group and voted for ‘John Doe’, or 5–10\% of the rows in the same age group vote for the ‘Jane Smith’. It may seem trivial to reason about such simple constraints with a COUNT query but the problem becomes much more difficult when constraints overlap and interact. For example, we may additionally about the rough income distributions in the polling population \(50k \leq \text{income} \leq 100k\) group. The addition of this constraint forces us to reason about the rows that could be in the conjunction \(18 \leq \text{age} \leq 25 \land (50k \leq \text{income} \leq 100k)\). PCs allow the user to specify and record such constraints and we construct a range of possible values the polling result could take given these constraints.
2.2 Scope and Implications

In this work, we focus on aggregate queries. All of our queries are of the form:

```sql
SELECT agg(attr)
FROM R1, ..., RN
WHERE ....
GROUP BY ....
```

We consider \texttt{SUM}, \texttt{COUNT}, \texttt{AVG}, \texttt{MIN}, and \texttt{MAX} aggregates with predicates and possible inner join conditions. Each of the tables $R_1, ..., R_n$ has a known “certain” partition denoted by $R_1^*, ..., R_n^*$ and unknown “missing” partition $R_1^?, ..., R_n^?$, where $R_i = R_i^* \cup R_i^?$. The user defines a set of constraints $\pi_1, ..., \pi_n$ over the possible values that each $R_i^?$ could take. The PC framework produces a , namely, the min and max value that one of the aggregate queries above can take given all of the data that are possible under the constraints.

*Predicate Constraints are formal descriptions of assumptions.* Our main motivation is that data scientists can fail to record their precise assumptions when making decisions with missing data. The predicate-constraint formalism enforces that there are testable constraints that are recorded during the decision making process. These constraints can be compiled into hard guarantees on aggregate query results. The optimization problem that translates these constraints into guarantees is not trivial when the constraints interact. Most of this paper will describe this optimization problem and how to run it at scales that are interesting to the database community.

*Predicate Constraints are a sparse generalization of histograms.* A histogram can be considered a 1-D “dense” predicate constraint which defines a partition of an attribute (a predicate) and the fraction of rows in that partition. Instead of a partition of a single attribute, a predicate constraint defines a general logical predicate and the attribute ranges of rows contained in that predicate. The predicates could possibly be overlapping, can correlate multiple attributes, and do not necessarily have to partition an entire domain. The analogy
to histograms is useful because the same way that histograms are employed in selectivity estimation, in lieu of sampling, because they are far less sensitive to highly selective queries, we believe that predicate-constraints offer the same advantages over statistical approaches in calculations.

*Predicate constraints have predictable failure modes.* We consider a “failure event” to be when a realized value is outside of a returned confidence interval. Statistical techniques provide probabilistic confidence intervals. For example, in a 99% statistical confidence interval there is a 1% chance that the values lie outside the confidence interval. Not only is there a random chance 1% of failure, sometimes the actually realized failures can be far higher if certain technical assumptions are violated. Result ranges derived from predicate-constraints only fail if the constraints themselves are violated (or specified incorrectly)—there are no random failures.
CHAPTER 3
APPROACH

3.1 Predicate-Constraints

This section describes our language for expressing constraints. We start our study considering a single unknown relation $R_i^2$. Suppose this unknown relation is over the attributes $A = \{a_1, ..., a_p\}$. The domain of each attribute $a_i$ is a set denoted by $\text{dom}(a_i)$. A row $r$ is an element drawn from the domain

$$D = \text{dom}(a_1) \times ... \times \text{dom}(a_p).$$

3.1.1 Predicate-Constraint

Each predicate-constraint models the values that rows that satisfy the predicate can take on as well as how frequently they could occur.

**Predicate:** A predicate $\psi$ is a Boolean function that maps each possible rows to a True and False value $\psi: D \mapsto \mathbb{Z}_2$. For efficient implementation, we focus on predicates that are conjunctions of ranges and inequalities. This restriction simplifies satisfiability testing, which is an important step in our algorithms.

**Value Constraint:** A value constraint specifies a set of ranges that each attribute can take on. A range of the attribute $a_i$ is defined as a row of two elements $(l, h) \in \text{dom}(a_i)$ where $l \leq h$. A value constraint $\nu$ is a set of ranges for each of the $p$ attributes:

$$\nu = \{(l_1, h_1), ..., (l_p, h_p)\}$$

$\nu$ defines a Boolean function as above $\nu: D \mapsto \mathbb{Z}_2$ that checks whether a row satisfies all the specified ranges. Since we focus on bounding aggregates it is sufficient to assume that the attribute ranges are over numerical attributes.
Frequency Constraint: Associated with each predicate is additionally a frequency constraint. This bounds the number of times that rows with the predicate appear. The frequency constraint is denoted as $\kappa = (k_l, k_u)$.

Predicate Constraint: A predicate-constraint is a three-tuple of these constituent pieces $\pi = (\psi, \nu, \kappa)$, a predicate, a set of value constraints, and a frequency constraint. The goal of $\pi$ is to define constraints on relations that satisfy the above schema. Essentially a predicate constraint says if $R$ is a relational instance that satisfies the predicate-constraint $\pi$ “For all rows that satisfy the predicate $\psi$, the values are bounded by $\nu$ and the number of such rows is bounded by $\kappa$”. Formally, we get the definition below.

[Predicate Constraint] A predicate constraint is a three-tuple consisting of a predicate, a value constraint, and a frequency constraint $\pi = (\psi, \nu, \kappa)$. Let $R$ be a relational instance over the attributes $A$. $R$ satisfies a predicate constraint denoted by $R \models \pi$ if:

$$
(\forall r \in R : \psi(r) \implies \nu(r)) \land k_l \leq |\{\forall r \in R : \psi(r)\}| \leq k_u
$$

Let us now consider several examples of predicate constraints using the election data example in the previous section on the 20 missing rows of the Vote table. Suppose, the data analyst wants to see what would happen if there are between 2 and 5 people in the age range who vote for 'John Doe' and 1–3 people in that range that vote for 'Jane Smith':

$$
c1 : (18 \leq \text{age} \leq 25) \implies \text{cand} = \text{`John Doe'}, \ (2, 5)
$$

$$
c2 : (18 \leq \text{age} \leq 25) \implies \text{cand} = \text{`Jane Smith'}, \ (1, 3)
$$

Suppose one wanted to define a simple partition over a single attribute, like the number of participants in the 18-25 age group. This can be done using a tautology. The following constraints enforces that there are between 1 and 4 people in the 18–25 age group and
between 10 and 20 people in the 26–35 age group:

\[ c3 : (18 \leq \text{age} \leq 25) \Rightarrow (18 \leq \text{age} \leq 25), (1, 4) \]

\[ c4 : (26 \leq \text{age} \leq 35) \Rightarrow (26 \leq \text{age} \leq 35), (10, 20) \]

We can similarly set global constraints to bound the total number of missing rows:

\[ c5 : True \Rightarrow *, (20, 20) \]

Observe how \( c3 \) interacts with \( c1 \) and \( c2 \). Some of the logical “worlds” allowed by \( c3 \) are disallowed by the other constraints. This interaction will be the main source of difficulty in computing based on a set of PCs.

### 3.1.2 Predicate-Constraint Sets

Users specify their assumptions about missing data using a set of predicate constraints \( A \). A predicate-constraint set is defined as follows:

\[ S = \{\pi_1, ..., \pi_n\} \]

\( S \) gives us enough information to bound the results of common aggregate queries when there is closure: every possible missing row satisfies at least one of the predicates. \( [\text{Closure}] \) Let \( S \) be a predicate constraint set with the elements \( \pi_i = (\psi_i, \nu_i, \kappa_i) \). \( S \) is closed over an attribute domain \( D \) if for every \( t \in D \):

\[ \exists \pi_i \in S : \psi_i(t) \]

Closure is akin to the traditional closed world assumption in logical systems, namely, the predicate constraints completely characterize the behavior of the missing rows over the domain.
3.1.3 Discussion

It is important to note, that our objective is not to build the most expressive language to represent uncertain data but rather one that we can pragmatically use to bound aggregate queries.

**c-tables [12]:** c-tables are one of the classical approaches for representing missing data in a relation. The relationship with Predicate-Constraint sets is nuanced. Due to the frequency constraints in Predicate-Constraint sets, we can represent cases that go beyond the typical closed-world assumption (CWA) is required in c-tables, where all records are known in advance and null cells are specifically annotated. However, the constraint language in c-tables is more expressive (what values the missing cells can take on). So under the CWA, Predicate-Constraint sets are strictly less expressive, however we can express a limited number of open-world models.

**m-Tables [30]:** m-tables extend c-tables to include variable cardinality representations to go beyond the CWA. In m-tables, cardinality constraints are specified per-relation. We specify frequency constraints per predicate. However, like c-tables, the actual value constraints in m-tables are more expressive. Our goal is to be able to bound SQL aggregate queries so highly expressive constraints are hard to reason about. Sundarmurthy et al. [30] do not consider the problem of aggregate query processing on the uncertain relations.

**Probabilistic Databases:** While predicate-constraint sets model uncertainty, they are not a probabilistic model for uncertainty as in probabilistic databases [28]. Our constraint language is fully deterministic.
4.1 Calculating

This section focuses on a simplified version of the bounding problem. We consider a single table and single attribute aggregates. Let \( q \) denote such an aggregate query. The problem to calculate the upper bound is:

\[
 u = \max_{R} q(R) \quad (4.1)
\]

subject to: \( R \models S \)

We will show that our bounds are tight—meaning that the bound found by the optimization problem is a valid relation that satisfies the constraints.

Throughout the rest of the paper, we only consider the maximal problem. Unless otherwise noted, our approach also solves the lower bound problem:

\[
 l = \min_{R \in \mathcal{R}} q(R)
\]

subject to: \( R \models S \)

Specifically, we solve the lower bound problem in two settings. In a general setting where there is no additional constraint, the minimal problem is can be solved by maximizing the negated problem: we first negate the value constraints, solve the maximizing problem with negated weights, and negate the final result.

In a special but also common setting, all the frequency constraints’ lower bounds are 0 (i.e., each relation has no minimum number of missing rows) and the value constraints’ lower bounds are 0 (i.e., all attributes are non-negative), the lower bound is trivially attained by the absent of missing row.
Figure 4.1: Predicates in a predicate-constraint set are possibly overlapping. The first step of the algorithm is to decompose a set of predicate-constraints into disjoint cells.

4.1.1 Cell Decomposition

Intuitively, we can think about the optimization problem as an allocation. We strategically assign rows in each of the predicates to maximize the aggregate query. However, the first challenge is that a row may fall in multiple Predicate-Constraints’ predicates, so this row may “count towards” multiple Predicate-Constraints. As the first step of the solution, we decompose the potential overlapping Predicate-Constraints’ predicates into disjoint cells.

An example that illustrates the decomposition is depicted in Figure 4.1. Each predicate represents a sub-domain. For each subset of predicates, a cell is a domain that only belongs to these predicates and not others. Thus, for $n$ predicates in a predicate-constraint set there are at most $O(2^n)$ cells. The cells take the form of conjunctions and negations of the predicates of each of the $n$ predicate constraints:

$$c_0 = \psi_1 \land \ldots \land \psi_n$$

$$c_1 = \psi_1 \land \ldots \land \neg \psi_n$$

$$c_2 = \psi_1 \land \ldots \land \neg \psi_{n-1} \land \psi_n$$
For each $c_i$, we have to reconcile the active predicate constraints (not negated above). Each cell is thus assigned the most restrictive upper and lower value bounds, and upper and lower cardinality bounds in the set of active constraints. Not all possible cells will be satisfiable—where there exists a row $t \in \mathcal{D}$ that satisfies the new predicate-constraint. As in Figure 4.1, there are 7 possible subsets, but there are only 5 satisfiable cells. We use the Z3 \cite{9} solver to prune all the cells that are not satisfiable.

Optimization 1. Predicate Pushdown: Cell decomposition is a costly process for two reasons. First, there is a potentially exponential number of cells. Second, determining whether each cell is satisfiable is not always easy (each check is on the order of 10’s of ms). One obvious optimization is to push down the predicates of the target query into the decomposition process. When the target query has predicates, we exclude all cells that do not overlap with the query’s predicate.

Optimization 2. DFS Pruning: The naive solution is to simply sequentially iterate through all of the cells and test each for satisfiability. Note that for a problem with $n$ PCs, each logical expression describing a cell is a conjunction of $n$ predicates. Conjunctions can be short-circuited if any of their constituent elements evaluate to false. Therefore, the process of evaluating the satisfiability of the cells can be improved by using a Depth First Search that evaluates prefixes of growing length rather than a sequential evaluation. With DFS, we can start from the root node of all expressions of length 1, add new PCs to the expression as we traverse deeper until we reach the leaf nodes which represent expressions of length $n$. As we traverse the tree, if a sub-expression is verified by Z3 to be unsatisfiable, then we can perform pruning because any expression contains the sub-expression is unsatisfiable.

Optimization 3. Expression Re-writing: To further improve the DFS process, we can re-write the logical expressions to have an even higher pruning rate. There is one simple
re-writing heuristic we apply:

\[(X \land \neg(X \land Y)) = True \implies X \land \neg Y = True\]

It means if we have verified a sub-expression \(X\) to be satisfiable, and we also verified that after adding a new PC \(Y\), the expression \(X \land Y\) becomes unsatisfiable. We can conclude that \(X \land \neg Y\) is satisfiable without calling Z3 to verify. As shown in the experiment section, the DFS pruning technique combined with the rewriting can prune over 99.9% cells in real-world problems.

Optimization 4. Approximate Early Stopping: With the DFS pruning technique, we always get the correct cell decomposition result because we only prune cells that are verified as unsatisfiable. We also propose an approximation that can trade range tightness for a decreased run time. The idea is to introduce ‘False-Positives’, i.e., after we have used DFS to handle the first \(K\) layers (sub-expressions of size \(K\)), we stop the verification and consider all cells that have not been pruned as satisfiable. These cells are then admitted to the next phase of processing where we use them to formulate a problem that can be solved to get our bound. Admitting unsatisfiable cells introduces ‘False-Positives’ that would make our bound loose, but it will not violate the correctness of the result (i.e. the result is still a bound) because: (1) the ‘true-problem’ with correctly verified cells is now a sub-problem of the approximation and (2) the false-positive cells does not add new constraints to the ‘true-problem’.

\[4.1.2 \text{ Integer-Linear Program}\]

We assume that the cells in \(C\) are ordered in some way and indexed by \(i\). Based on the cell decomposition, we denote \(C_i\) as the (sub-)set of Predicate-Constraints that cover the cell \(i\). Then for each cell \(i\), we can define its maximal feasible value \(U_i(a) = \min_{p \in C_i} p.\nu.h_a\), i.e., the minimum of all \(C_i\)’s value constraints’ upper bounds on attribute \(a\).
A single general optimization program can be used to bound all the aggregates of interest. Suppose we are interested in the \textbf{SUM} over attribute \( a \), we slightly abuse the terms and define a vector \( U \) where \( U_i = U_i(a) \). Then, we can define another vector \( X \) which represents the decision variable. Each component is an integer that represents how many rows are allocated to the cell.

The optimization problem is as follows. To calculate the upper bound:

\[
\max_X U^T X
\]

subject to: \( \forall j, k_l^{(j)} \leq \sum_{i, j \in C_i} X[i] \leq k_u^{(j)} \)

\( \forall i \), \( X[i] \) is integer

As an example, the first constraint in Figure 4.1 is \( k_l^{1} \leq x_1 + x_2 \leq k_u^{1} \) for \( \pi_1 \), so on and so forth.

Given the output of the above optimization problem, we can get bounds on the following aggregates:

\textbf{COUNT:} The count of cardinality can be calculated by setting \( X \) as the unit vector (\( X_i = 1 \) for all \( i \)).

\textbf{AVG:} We binary search the average result: to testify whether \( r \) is a feasible average, we disallow all rows to take values smaller than \( r \) and invoke the above solution for the maximum \textbf{SUM} and the corresponding \textbf{COUNT}. If the overall average is above \( r \), then we test \( r' > r \), otherwise we continue testing \( r' < r \).

\textbf{MAX/MIN:} Assuming all cells are feasible, the max is the largest of all cells’ upper bound. Min can be handled in a similar way.
4.1.3 Complexity

This MILP can be solved with a commercial optimization solver such as Gurobi or CPLEX. This is a pragmatic approach to what is a computationally hard problem:

Determining the maximal sum of a relation constrained by a predicate-constraint is NP-Hard.

Sketch. We prove this proposition by reduction to the maximal independent set problem, which is NP-Hard. An independent set is a set of vertices in a graph, no two of which are adjacent. We can show that every independent set problem can be described as a predicate-constraint maximal sum problem. Let \( G = (V, E) \) be a graph. We can think of this graph as a relational table of “vertex” rows. For example: \((V1, V2)\). For each vertex \( v \in V \), we define a single Predicate-Constraint with a max value of 1 and a max frequency of 1 \((x = v, [0, 1], [0, 1])\). For each edge between \( v \) and \( v' \), we define another predicate constraint that is equality to either of the vertices also with a max frequency of 1:

\[(x = v \lor x = v', [0, 1], [0, 1]).\]

This predicate constraint exactly contains the two vertex constraints. Therefore, the cells after the cell-decomposition step perfectly align with those vertices. The optimization problem allocates rows to each of those cells but since the edge constraint has a max frequency is 1 only one of the vertex cells can get an allocation. Since all of the vertex constraints have the same max value, the optimization problem finds the most number of such allocations that are consistent along the edges (no two neighboring vertices both are allocated). This is exactly the definition of a maximal independent set. Since the maximal sum problem is more expressive than the maximal independent set its complexity is greater than NP-Hard in the number of cells. \( \square \)
4.1.4 Special Cases and Discussion

The hardness of the general problem should not discourage us because there are several redeeming aspects.

General Optimality An optimal solution to the MILP problem in equation 2, is guaranteed to be the tightest possible upper bound given the available information. There is a certificate of this tightness in the form of the \( \text{arg max} \) which is interpreted as a configuration of missing rows that actually achieves that upper bound (i.e., forces the aggregate to have that max value). There are, however, a couple of caveats. (1) If you do not apply the exact cell decomposition algorithm with satisfiability testing, the calculated bound might be loose. The satisfiability test prunes cells that should not receive an allocation in the above optimization algorithm, extra cells give the algorithm more slack to increase the bound. (2) If you apply the approximate decomposition algorithm, a similar problem happens where the cardinalities of inexact decomposed cells are higher than they should be.

Faster Algorithm in Special Cases In the case that the Predicate-Constraints have disjoint predicates, the cell decomposition problem becomes trivial as each predicate is a cell. The MILP problem also degenerates since all the constraints, besides each variable being integer, do not exist at all. Thus, if we take the \text{SUM} problem as an example, the solution is simply the sum of each Predicate-Constraints’ maximum sum, which is the product of the maximum value and the maximum cardinality. This disjoint case is related to the work in data privacy by Zhang et al. [32].
CHAPTER 5

EXPERIMENTS AND EVALUATIONS

Now, we evaluate the Predicate-Constraint framework to demonstrate its utility in real-world scenarios. Our framework produces hard guarantees under the user-stipulated constraints. The main concern would be that the calculated range is too loose to be actionable, however, we show that this is not the case and it is often comparable (or better) than probabilistic confidence intervals produced by statistical inference frameworks.

5.1 Experimental Setup

We set up each experiment as follows: (1) summarize a dataset with N Predicate-Constraints, (2) each competitor framework gets a similar amount of information about the dataset (e.g., a statistical model with $O(N)$ statistical parameters), (3) we estimate a query result using each framework with calibrated error bounds.

In these evaluations, we leave it to the reader to decide how feasible getting such information is. The point of comparison is the contingency analysis given the amount of information provided about the missing rows: the range of values that the estimation framework says are possible for the aggregate query result. We evaluate the failure rate and the tightness of the estimated ranges v.s. the actual ranges for randomly generated queries.

Our comparative evaluation focuses on SUM, COUNT queries as those are what the baselines support. We show similar results on MIN, MAX, AVG queries\footnote{1. AVG queries with predicates are noted to be a subtle challenge in the approximate query processing literature [18].} but only within our framework.
5.1.1 Sampling

The first class of techniques that we consider are sampling techniques. We assume that the user provides example missing data records—again this might be arguably more difficult than simply describing the attribute ranges as in a predicate constraint—but we are generous to the baseline. We use only these examples to extrapolate a range of values that the missing rows could take.

**Uniform Sampling** We randomly draw $N$ samples (US-1) and $10N$ samples (US-10) from the set of missing rows.

**Stratified Sampling** We also use a stratified sampling method which performs a weighted sampling from partitions defined by the PCs that we use for a given problem. Similarly, we denote $N$ samples as (ST-1) and $10N$ samples (ST-10).

For both of these sampling schemes, there is a question of how to calculate a confidence interval. Commonly, approximate query processing uses the Central Limit Theorem to derive bounds. We apply the formulas described in [18] to formulate 99% confidence intervals. These confidence intervals are parametric as they assume that the error in estimation is Normally distributed, and we denote these estimates as US-1p, ST-10p, etc. Alternatively, one could use a non-parametric method, which does not make the Normal assumption, to estimate confidence intervals like those described in [11]. Likewise, we denote these as US-1n, ST-10n, etc.

5.1.2 Generative Model

Another approach is to fit a generative model to the missing data. We use the missing data as training data for a Gaussian Mixture Model (GMM). The trained GMM is used to generate data of the size of the original missing rows. A query result that is evaluated on the generated data is returned. This is simulating a scenario where there is a generative model that can describe what data is missing. If we run this process several times, we can
determine a range of likely values that the query could take.

5.1.3 Equiwidth Histogram

We build a histogram on all of the missing data on the aggregate attribute with \( N \) buckets and use it to answer queries. We use standard independence assumptions to handle queries across multiple attributes.

5.1.4 PCs

The accuracy of the PC framework is dependent on the particular PCs that are used for a given task. In our “macro-benchmarks”, we try to rule out the effects of overly tuned PCs. We consider two general schemes: Corr-PC, even partitions of attributes correlated with the aggregate of interest, and Rand-PC, randomly generated PCs.

For Corr-PC, we identify the (other than the aggregation attribute) most correlated with the aggregate to partition on. We divide the combined space into equi-cardinality buckets where each partition contains roughly the same number of tuples. We omit the details of this scheme for the sake of brevity. For Rand-PC, we generate random overlapping predicate constraints over the same attributes (not necessarily equi-cardinality). We take extra care to ensure they adequately cover the space to be able to answer the desired queries. We see these as two extremes: the reasonably best performance one could expect out of the PC framework and the worst performance. We envision that natural use cases where a user is manually defining PCs will be somewhere in the middle in terms of performance.

5.1.5 Datasets

We evaluate using the following datasets.

Intel Wireless Dataset The Intel wireless dataset [24] contains data collected from 54 sensors deployed in the Intel Berkeley Research lab in 2004. This dataset contains 8 columns
including different measurements like humidity, temperature, light, voltage as well as date and time of each record. For this dataset, Corr-PC and Rand-PC are defined as $N = 100$ constraints over device_id and time.

Border Crossing Dataset The Border Crossing dataset [22] from The Bureau of Transportation Statistics (BTS) contains summary statistics for inbound crossings at the U.S.-Canada and the U.S.-Mexico border at the port level. This dataset contains 8 columns that describe the summary of border crossing (type of vehicles and the count) that happen at a port (port code, port location, state, etc) on a specific date. We compare the hard bound baselines with PCs using different partitions with three groups of randomly generated queries. Corr-PC and Rand-PC are defined as $N = 100$ constraints over port and date.

Airbnb Dataset The Airbnb dataset [2] contains open data from Airbnb listings and metrics in the city of New York, 2019. This dataset contains 19 columns that describe different properties of listings like location (latitude, longitude), type of room, price, number of reviews, etc. Corr-PC and Rand-PC are defined as $N = 50$ constraints over latitude.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Query</th>
<th>PredAttr</th>
<th>PC</th>
<th>Hist</th>
<th>US-1p</th>
<th>US-10p</th>
<th>US-1n</th>
<th>US-10n</th>
<th>ST-1n</th>
<th>ST-10n</th>
<th>Gen</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>COUNT(*)</td>
<td>DevID</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>53</td>
</tr>
<tr>
<td>Intel Wireless</td>
<td>DevID, Time</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>SUM(Light)</td>
<td>DevID</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>DevID, Time</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>2</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Latitude</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>COUNT(*)</td>
<td>Longitude</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Lat, Lon</td>
<td>0</td>
<td>0</td>
<td>23</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Airbnb@NYC</td>
<td>Latitude</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>40</td>
<td>5</td>
<td>0</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SUM(Price)</td>
<td>Longitude</td>
<td>0</td>
<td>0</td>
<td>22</td>
<td>2</td>
<td>16</td>
<td>5</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lat, Lon</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td>13</td>
<td>27</td>
<td>5</td>
<td>32</td>
<td>8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Port</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>37</td>
</tr>
<tr>
<td>COUNT(*)</td>
<td>Date</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Port, Date</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>Border Cross</td>
<td>Port</td>
<td>0</td>
<td>0</td>
<td>91</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SUM(Value)</td>
<td>Date</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>10</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Port, Date</td>
<td>0</td>
<td>0</td>
<td>54</td>
<td>26</td>
<td>29</td>
<td>1</td>
<td>20</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Over a 100 randomly chosen predicates, we record the failure rate in percentage of different “error bound” frameworks. A failure event is one where an observed outcome is outside the range of the returned bounds. For a 99% confidence interval, as used in our experiments, one would expect a 1% failure rate for the sampling-based frameworks—but is significantly higher in practice for small skewed datasets. PCs, and as a special case Histograms, are guaranteed not to fail if the assumptions are satisfied.
5.2 Probabilistic Confidence Intervals are Often Unreliable on Real Data

The most common way to assess one’s confidence in a query result is to use a probabilistic confidence interval technique. Table 5.1 presents different techniques and their “failure rate” over 100 queries, which is the number of queries for which the true value exceeded what was produced in a bound. The most common technique by far is to rely on the Central Limit Theorem (US-1p, US-10p). Estimating this standard error from a sample is often far more unreliable than one would normally expect. We use a 99% confidence interval for a CLT bound given $N$ samples and $10N$ samples, and observe that the failure rate is far higher than 1%. In this missing data setting, a small number of example rows fail to accurately capture the “spread” of a distribution.

Next, we can make the sample-based confidence intervals a much more conservative non-parametric model (US-1n, US-10n), which holds under milder assumptions. Such a bound relies on an estimate of the min and max values and not an accurate estimate of the standard error. Predictably, this performs much better than the CLT approach. However, as we can see in the table, non-parametric bound baselines still fail more often than one would expect over 100 queries. Small samples and selective queries pose a fundamental challenge to these approaches. Stratified samples do not solve this problem either. While, they cover the space more evenly, for any given strata, they can have a high failure rate.

One could intuitively fix this problem by annotating the strata in a stratified sample with metadata that accurately depicts min and max values. This is exactly the definition of PCs. The PC technique and Histograms always generate hard bounds for queries because for the same number of “bits” of information they capture the entire spread of values much more accurately. For the purposes of bounding, the example tuples provided by a sample are not as useful as the ranges.

Finally, we use the generative approach to model the joint data distribution. We draw
samples from this model and use that to produce a confidence interval. Such an approach works very well on some datasets/queries but not others. These experiments illustrate how important a guaranteed “0 failure rate” is for real-world decision making. Statistical confidence intervals can give a false sense of security in real-world data.

5.3 PCs are Competitively Accurate

We show that the bounds calculated using a PC are often of similar accuracy to probabilistic techniques. We show results with both good (Corr-PC) and random choices of predicates (Rand-PC).

We use the same queries defined in the previous experiment. We compare the actual query result to the predicted bound. Quantitatively, we present this as an upper bound result ratio between the upper bound and the result. We present the median of this ratio over a 100 queries with random predicates and term this the “over-estimation” rate. Of course, this is only meaningful when the upper bound has a low failure rate—so we compare PCs to Histograms and US-10n. Lower is better on these results.

5.3.1 Intel Wireless Dataset

Figure 5.1 presents the results for SUM and COUNT queries on the Intel Wireless dataset. Both well-informed PCs (Corr-PC) and randomly chosen PCs (Rand-PC) have similar accuracy to the sampling baseline (despite a 10x less data footprint). Of course, Rand-PC over-estimates the actual value more significantly. The result also shows that a well-designed PC is significantly more effective than a histogram. While sampling is very accurate on average across many queries, PCs are guaranteed to have 0 failures. Despite this guarantee, the bounds are not vacuous and within an order of magnitude of those produced by a 10x data sample.
5.3.2 Airbnb at New York City Dataset

Figure 5.2 replicates the same experiment on a different dataset. This dataset is significantly skewed compared to the Intel Wireless dataset, so the estimates are naturally harder to produce. As before, we find that well-designed PCs are just as tight as sampling-based bounds. However, randomly chosen PCs are significantly looser (more than 10x). PCs fail conservatively, a loose bound is still a bound, it might just not be that informative. In skewed data such as this one, we advise that users design simple PCs that are more similar to histograms (partition the skewed attribute).

5.3.3 Border Crossing Dataset

Results in Figure 5.3 show results on another skewed dataset. As before, informed PCs are very accurate (in fact more accurate than sampling). Randomly chosen PCs over-estimate the result range by about 10x compared to the other approaches. Again, the advantage of the PC framework is that unless the assumptions are violated, there are no random failures.
On this dataset, over 100 queries, we observed one bound failure for the sampling approach. This failure is included in the results.

5.3.4 PCs are Data Efficient

In all of our experiments above, we consider a 10x random sample. Where the baseline is given 10 times the amount of data compared to the number of PCs. In Figure 5.4, we use the Intel Wireless dataset to demonstrate the performance of parametric sampling bounds and the non-parametric bounds using different sample sizes. And the results demonstrate a clear trend of convergence as we increase the sample size. If we consider data parity (1x), a non-parametric bound is significantly less accurate than a well-designed PC.

5.4 Worst-Case Optimal Estimates On Joins

The statistical approaches do not generalize well to estimates for queries with inner equal joins, and we found the bounds produced were too fallible for meaningful comparison. To evaluate PCs on such queries, we compare to another class of bounding techniques that have
been proposed in the privacy literature. These bounds estimate how much a query might change for a single hypothetical point update. Our insight connecting the bounding problem to worst-case optimal join results leads to far tighter bounds in those settings. Johnson et al. [13] proposed a technique named elastic sensitivity that can bound the maximum difference between the query’s result on two instances of a database.

5.4.1 Counting Triangles

In this example, which is also studied by Johnson et al. [13], we analyze a query that is used to count triangles in a directed graph. We generate a hypothetical dataset and measure the performance. In Figure 5.5 (TOP), we show the results of the two approaches on the counting triangle problem using randomly populated edges tables of different sizes. And the results confirm that our approach drives a bound that is much tighter in this case—in fact by multiple orders of magnitude.
5.4.2 Acyclic Joins

We also consider the following join query: $R_1(x_1, x_2) \Join \Join R_2(x_2, x_3) \Join \ldots \Join R_5(x_5, x_6)$ We generate 5 tables, each with $K$ rows and use the two approaches to evaluate the size of the join results. We vary the value of $K$ to understand how the bounds change accordingly. The results are shown in Figure 5.5 (BOTTOM), we can see that elastic sensitivity always assumes the worst-case scenario thus generates the bound for a Cartesian product of the tables that is several magnitudes looser than our approach.

5.5 Optimizations for Overlapping PCs

In this section, we evaluate the scalability of PCs and optimization techniques. As mentioned in earlier sections, the complete process of solving a PC problem including two parts. First, we need to perform cell decomposition to find out which cells are valid and second, we need to formalize a MILP problem with the valid cells that can be solved to find the optimal bound. The vanilla version described above is intractable because the number of sub-problems we need to solve in the cell decomposition phase is exponential to the number of PCs.
Figure 5.5: We compare the bound derived by our approach (Corr-PC) with state of the art baseline Elastic Sensitivity on the triangle counting problem of different table sizes (TOP) and an acyclic join (BOTTOM).
We presented a number of optimizations in the paper to improve this time. We will show that naive processing of the PCs leads to impractical running time. We generate 20 random PCs that are very significantly overlapping. Figure 5.6 plots the number of cells evaluated as well as the run time of the process. The naive algorithm evaluates the SAT solver on more than 1000x more cells than our optimized approach.

Since cell decomposition is really the most expensive step We can prune and save about 99.9% of the solving time by using DFS (early termination for cell decomposition) and the rewriting heuristic. Without these optimizations, PCs are simply impractical at large scales.

5.6 Non-Overlapping PCs Scale Effectively

PCs can be solved significantly faster in the special case of partitioned PCs (non-overlapping). The process of answering a query with PC partitions is much simpler than using overlapping PCs. Because partitions are disjoint with each other, we can skip the cell decomposition, and the optimization problem can be solved by a greedy algorithm. As shown in Figure 5.7, the average time cost to solve one query with a partition of size 2000 is 50ms, and the time
cost is linear to the partition size. We can scale up to 1000s of PCs in this case.

5.7 PCs Are Relatively Robust To Noise

We introduce noise into the PCs to understand how incorrectly defined PCs affect the estimated ranges. When the PCs are noisy, there are failures because the ranges could be incorrect. Figure 5.8 plots the results for Corr-PC, a set of 10 overlapping PCs (Overlapping-PC), and US-10n. We corrupt the sampling bound by mis-estimating the spread of values. All experiments are on the SUM query for the Intel Wireless dataset as in our previous experiment. For corrupting noise that is drawn from a Gaussian distribution of 1, 2 and 3 standard deviation, we plot the failure rate.

US-10n has the greatest increase in failures due to the noise. Corr-PC is significantly more robust. Finally, Overlapping-PC is the most robust. This experiment illustrates the benefits of overlapping PCs. When one such overlapping PC is incorrect, our framework automatically applies the most restrictive overlapping component. This allows the framework to reject some amount of mis-specification errors.
5.8 PCs Are Not Sensitive to Query Selectivity

In this experiment, we generate random queries with different selectivity to understand how selectivity would affect the bound derived by PC. We generate target queries that contains 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% of the population and the results are shown in Figure 5.9. The results suggest that as the selectivity of query increases, the bound generated by PC will be tighter, this coordinate with our expectation because higher selectivity generally leads to a smaller gap between ground truth and the bound because more rows are included in the query result.

5.9 Handling MIN, MAX, and AVG Queries

As mentioned in earlier sections, besides COUNT and SUM queries, PCs can also handle MIN, MAX and AVG queries. In this experiment, we use the Intel Wireless dataset for demonstration, we partition the dataset on DeviceID and Time. For each type of query, we randomly generate 100 queries and use PC to solve them. Results can be found in Figure
5.10. First, note how PC can always generate the optimal bound for MIN and MAX queries. PCs are a very good representation of the spread of the data, more so than a sample.

We show similar performance for AVG queries to the COUNT and SUM queries studied before. AVG queries are an interesting case that we chose not to focus on. While sampling is a very accurate estimate of AVG queries without predicates, with predicates the story becomes more complicated. Since averages are normalized by the number of records that satisfy the predicate, you get a "ratio of estimators" problem and the estimate is not exactly unbiased. So for small sample sizes, standard bounding approaches can have a high failure rate despite seemingly accurate average-case performance.
Figure 5.10: With PC, an optimal bound can be derived for MIN and MAX queries. PC also generates competitive result for AVG Queries.
CHAPTER 6
DISCUSSION AND RELATED WORK

6.1 Related Work

The overarching challenge addressed in the PC framework is related to the concept of “reverse
data management” proposed by Meliou et al. [19, 20]. Meliou et al. argue that as data grow
in complexity, analysts will increasingly want to know not what their data currently says
but what changes have to happen to the dataset to force a certain outcome. Such how-to
analyses are useful in debugging, understanding sensitivity, as well as planning for future
data. Meliou et al. build on a long line of what-if analysis and data provenance research,
which study simulating hypothetical updates to a database and understanding how query
results might change [10, 4]. While we address similar concerns to this line of work in
spirit, our focus on aggregate queries and confidence intervals leads to a very different set of
technical contributions. The PC framework should be evaluated much more like a synopsis
data structure than a data provenance reasoning systems.

Therefore, our experiments largely focus on evaluations against other synopsis structures
and how to extract confidence intervals from them [8]. While Approximate Query Processing
(AQP) has been studied for several decades [23], it is well-known that the confidence inter-
vals produced can be hard to interpret [15]. This is because estimating the spread of high
dimensional data from a small sample is fundamentally hard, and the most commonly used
Central-Limit Theorem-based confidence intervals rely on estimated sample variance. Espe-
cially for selective queries, these estimates can be highly fallible—a 95% confidence interval
may “fail” significantly more than 5% of the time [1]. Unfortunately, as confidence intervals
become more conservative, e.g., using more general statistical bounding techniques, their
utility drops [11]. In a sense, our optimization algorithm automatically navigates this trade-
off. The algorithm optimizes the tightest bound given the available information in the form
of PCs. We interpret PCs as generalized histograms with overlapping buckets and uncertain
bucket counts. Despite these differences with AQP, we do believe that the connections between uncertainty estimation and dirty data (like missing rows) are under-studied [17, 18]. We also believe that in future work mixed systems with both PCs and samples can have the best of both worlds, e.g., augmenting Quickr with PC estimation [14].

Deterministic alternatives to AQP have been studied in some prior work. Potti et al. propose a paradigm called DAQ [26] that does reason about hard ranges instead of confidence intervals. DAQ models uncertainty at relation-level instead of predicate-level like in PCs and DAQ does not handle cardinality variation. In the limited scenario of windowed queries over time-series data, deterministic bounds have been studied [3]. The technical challenge arises with overlapping constraints and complex query structures (like join conditions and arbitrary predicates). Similarly, we believe that classical variance reduction techniques for histograms could be useful for PC generation in future work [25], since histograms are a dense 1-D special case of our work.

There is also a rich line of work that studies missing rows from databases. m-tables study variable cardinality representations to go beyond the CWA. In m-tables, cardinality constraints are specified per-relation. We specify frequency constraints per predicate. However, Sundarmurthy et al. [30] do not consider the problem of aggregate query processing on uncertain relations. There is similarly related work that studies intentionally withholding partitioned data for improved approximate query performance [31]. We believe that the novelty of our framework is the efficient estimation of aggregate query confidence intervals. Similarly, the work by Burdik et al. is highly related where they study databases with certain “imprecise” regions instead of realized tuples [5]. And the approach proposed by Cai et al. [6] based on random hash partitions can only handle cardinality estimations over joins. Cai et al. highlight many of the core challenges but fails to produce confidence intervals or handle inner-equality join queries optimally like our framework.

As seen in our experiments, the privacy literature has studied version of this bounded aggregate queries on uncertain data [32, 13]. In fact, Zhang et al. can be seen
as solving the partitioned version of our problem [32]. The privacy work had no reason to consider overlapping PCs and joins in the way that our work does.
CHAPTER 7
SUMMARY

We proposed a framework that can produce automatic contingency analysis, i.e., the range of values an aggregate SQL query could take, under formal constraints describing the variation and frequency of missing data tuples. There are several interesting avenues for future work. First, we are interested in studying these constraints in detail to model dirty or corrupted data. Rather than considering completely missing or dirty rows, we want to consider rows with some good and some faulty information. From a statistical inference perspective, this new problem statement likely constitutes a middle ground between sampling and Predicate-Constraints. Second, we would like to further understand the robustness properties of result ranges computed by Predicate-Constraints as well as other techniques. Understanding when result ranges are meaningful for real-world analytics will be an interesting study. Finally, we would like to extend the Predicate-Constraint framework to be more expressive and handle a broader set of queries.
REFERENCES


