SUMMARIZING HIGH-DIMENSIONAL FEATURE VECTORS

by

Ting Xie

March 2020

A dissertation submitted to the
Faculty of the Graduate School of
the University at Buffalo, State University of New York
in partial fulfilment of the requirements for the
degree of

Doctor of Philosophy

Department of Computer Science and Engineering

Dedication

To my lovely wife, Xiaojuan Xie, and my wonderful parents, Lixin Xie and Yun Ma, who have been selflessly helping me pursue my dreams.

Acknowledgments

I would like to express my deepest gratitude to

I wish to thank

My thanks also extend to

I want to thank all my former and present colleagues,

Last, but not least, I would like to thank my wife and my parents for their unconditional love and support.

Table of Contents

List of Tables

List of Figures

Abstract

todo

Chapter 1

Introduction

1.1 Motivation

Chapter 2

Application: Query Log Compression for Workload Analytics

2.1 Introduction

Automated analysis of database access logs is critical for solving a wide range of problems, from database performance tuning [2], to compliance validation [3], and query recommendation [4]. For example, the Peloton self-tuning database [5] searches for optimal configurations by repeatedly simulating database performance based on statistical properties of historical queries. Unfortunately, query logs for production databases can grow to be large — A recent study of queries at a major US bank for a period of 19 hours found nearly 17 million SQL queries and over 60 million stored procedure executions [6] — and computing these properties from the log itself is slow.

Tracking only a sample of these queries is not sufficient, as rare queries can disproportionately affect database performance, for example, if they benefit from an otherwise unnecessary index. Rather, we need a compressed *summary* of the log on

which we can compute aggregate statistical properties. The problems of compression and summarization have been studied extensively (e.g., [7, 8, 9, 10, 11, 12, 13, 14]). However, these schemes either require the use of heavyweight inference to desired statistical measures, or produce unnecessarily large encodings.

In this paper, we adapt ideas from pattern mining and summarization [15, 16] to propose a middle-ground: LogR, a summarization scheme that facilitates efficient (both in terms of storage and time) approximation of workload statistics. By adjusting a tunable parameter in LogR, users can choose to obtain a high-fidelity, albeit large summary, or obtain a more compact summary with lower fidelity. Constructing the summary that best balances compactness and fidelity is challenging, as the search space of candidate summaries is combinatorially large [15, 16]. LogR offers a new approach to summary construction that avoids searching this space, making inexpensive, accurate computation of aggregate workload statistics possible. As a secondary benefit, the resulting summaries are also human-interpretable.

LogR does not admit closed-form solutions to classical fidelity measures like information loss, so we propose an alternative called *Reproduction Error*. We show through a combination of analytical and experimental evidence that Reproduction Error is highly correlated with several classical measures of encoding fidelity.

LogR-compressed data relies on a codebook of structural elements like `SELECT` items, `FROM` tables, or conjunctive `WHERE` clauses [17]. This codebook provides a bi-directional mapping from SQL queries to a bit-vector encoding, reducing the compression problem to one of compactly encoding a collection of feature-vectors. We further simplify the problem by observing that a common theme in use cases like automated performance tuning or query recommendation is the need for predominantly aggregate workload statistics. As these are order-independent, we are able to focus exclusively on compactly representing *bags* of feature-vectors.

LogR works by identifying groups of co-occurring structural elements that we call patterns. We define a family of *pattern encodings* of access logs, which map patterns to their frequencies in the log. For pattern encodings, we consider two idealized measures of fidelity: (1) Ambiguity, which measures how much room the encoding leaves for interpretation; and (2) Deviation, which measures how reliably the encoding approximates the original log. Neither Ambiguity nor Deviation can be computed efficiently for pattern encodings. Hence we propose a measure called *Reproduction Error* that is efficiently computable and that closely tracks both Ambiguity and Deviation.

In general, the size of the encoding is inversely related with Reproduction Error: The more detailed the encoding, the more faithfully it represents the original log. Thus, log compression may be defined as a search over the space of pattern-based encodings to identify the one that best trades off between these two properties. Unfortunately, searching for such an ideal encoding from the space can be computationally expensive [15, 16]. To overcome this limitation, we reduce the search space by first clustering entries in the log and then encoding each cluster separately, an approach that we call *pattern mixture encoding*. Finally we identify a simple approach for encoding individual clusters that we call *naive mixture encodings*, and show experimentally that it produces results competitive with more general techniques for log compression and summarization.

Concretely, in this paper we make the following contributions: (1) We define two families of compression for query logs: pattern and pattern mixture, (2) We define a computationally efficient measure, Reproduction Error, and demonstrate that it is a close approximation of Ambiguity and Deviation (two commonly used measures), (3) We propose a clustering-based approach to efficiently search for naive mixture encodings, and show how these encodings can be further optimized, and, (4) We

experimentally validate LogR and show that it produces more precise encodings, faster than several state-of-the-art pattern encoding algorithms.

Roadmap. The paper is organized as follows: Section 2.2 formally defines the log compression problem and the summary representation; Section 2.3 then defines information loss of the summaries; Section 2.4 explains the difficulty in computing classical loss measures and provides a practical alternative; Section 2.5 motivates data partitioning and generalizes the practical loss measure to partitioned data; Section 2.6 then introduces the proposed LogR compression scheme; Section 2.7 empirically validates the practical loss measure and evaluates the effectiveness of LogR by comparing it with two state-of-the-art summarization methods; Section 2.8 further evaluates LogR under applications of comparison methods; Section 2.9 discusses related work. Section 3.8 concludes the paper and Section 2.11 discusses future work.

2.2 Problem Definition

In this section, we introduce and formally define the log compression problem. We begin by exploring several applications that need to repeatedly analyze query logs.

Index Selection. Selecting an appropriate set of indexes requires trading off between update costs, access costs, and limitations on available storage space. Existing strategies for selecting a near-optimal set of indexes typically repeatedly simulate database performance under different combinations of indexes, which in turn requires repeatedly estimating the frequency of specific predicates in the workload.

Materialized View Selection. The results of joins or highly selective selection predicates are good candidates for materialization when they appear frequently in the workload. Like index selection, view selection is a non-convex optimization problem,

typically requiring repeated simulation, which in turn requires repeated frequency estimation over the workload.

Online Database Monitoring. In production settings, it is common to monitor databases for atypical usage patterns that could indicate a serious bug or security threat. When query logs are monitored, it is often done retrospectively, some hours after-the-fact [6]. To support real-time monitoring it is necessary to quickly compute the frequency of a particular class of queries in the system's typical workload.

In each case, the application's interactions with the log amount to counting queries that have specific features: selection predicates, joins, or similar.

2.2.1 Preliminaries and Notation

Let L be a log, or a finite collection of queries $\mathbf{q} \in L$. We write $f \in \mathbf{q}$ to indicate that $\mathbf{q}$ has some *feature* f, such as a specific predicate or table in its `FROM` clause. We assume (1) that the universe of features in both a log and a query is enumerable and finite, (2) that the features are selected to suit specific applications and (3) optionally that a query is isomorphic to its feature set (motivated in Section 2.2.3.2). We outline one approach to extracting features that satisfies all three assumptions below. We abuse syntax and write $\mathbf{q}$ to denote both the query itself, as well as the set of its features.

Let ω denote some set of features $f \in \omega$. We write these sets using vector notation: $\omega = (x_1, \ldots, x_n)$ where n is the number of distinct features in the entire log and x_i indicates the presence (absence) of ith feature with a 1 (resp., 0). For any two patterns ω, ω', we say that ω' *is contained* or *appears* in ω if $\omega' \subseteq \omega$, or equivalently if $\forall i,\ x'_i \leq x_i$.

2.2.2 Coding Queries

For this paper, we specifically adopt the feature extraction conventions of a query summarization scheme by Aligon et al. [17]. In this scheme, each feature is one of the following three query elements: (1) a table or sub-query in the `FROM` clause, (2) a column in the `SELECT` clause, and (3) a conjunctive atom of the `WHERE` clause.

Example 1. *Consider the following example query.*

```
SELECT _id, sms_type, _time FROM Messages
WHERE status=? AND transport_type=?
```

The query has 6 *features*: ⟨ `_id`, `SELECT` ⟩, ⟨ `sms_type`, `SELECT` ⟩, ⟨ `_time`, `SELECT` ⟩, ⟨ `Messages`, `FROM` ⟩, ⟨ `status=?`, `WHERE` ⟩, *and* ⟨ `transport_type=?`, `WHERE` ⟩

Although this scheme is simple and limited to conjunctive queries, it fulfills all three assumptions we make on feature extraction schemes. The features of a query (and consequently a log) are enumerable and finite, and the feature set of the query is isomorphic to the original query. Furthermore, even if a query is not itself conjunctive, it may be rewritable into a conjunctive equivalent.

Although we do not explore more advanced feature encoding schemes in detail here, we direct the interested reader to work on query summarization [18, 19, 6]. For example, a scheme by Makiyama et. al. [18] also captures aggregation-related features like group-by columns, while an approach by Kul et. al. [6] encodes partial tree-structures in the query.

2.2.3 Log Compression

As a lossy form of compression, LogR only approximates the information content of a query log. We next develop a simplified form of LogR that we call pattern-based encoding, and develop a framework for reasoning about the fidelity of a LogR-compressed log. As a basis for this framework, we first formulate the information content of a query log to allow us to adapt classical measures of information content.

2.2.3.1 Information Content of Logs

We define the information content of the log as a distribution $p(Q \mid L)$ of queries Q drawn uniformly from the log.

Example 2. *Consider the following query log, which consists of four conjunctive queries.*

```
1. SELECT _id FROM Messages WHERE status = ?
2. SELECT _time FROM Messages
          WHERE status = ? AND sms_type = ?
3. SELECT _id FROM Messages WHERE status = ?
4. SELECT sms_type, _time FROM Messages
          WHERE sms_type = ?
```

Drawing uniformly from the log, each entry will appear with probability $\frac{1}{4} = 0.25$. *The query* q_1 $(= q_3)$ *occurs twice, so the probability of drawing it is double that of the others (i.e.,* $p(\mathbf{q}_1 \mid L) = p(\mathbf{q}_3 \mid L) = \frac{2}{4} = 0.5$*)*

Treating a query as a vector of its component features, we can define a query $\mathbf{q} = (x_1, \ldots, x_n)$ to be an observation of the multivariate distribution over variables $Q = (X_1, \ldots, X_n)$ corresponding to features. The event $X_i = 1$ occurs if feature i appears in a uniformly drawn query.

```
SELECT  sms_type,  external_ids,  _time,
        _id
  FROM  messages
 WHERE  (sms_type=?) ∧ (status=?)
```

(a) *Correlation-ignorant*: Features are highlighted independently

```
SELECT sms_type FROM messages WHERE sms_type=?
SELECT sms_type FROM messages WHERE status=?
```

(b) *Correlation-aware*: Pattern groups are highlighted together.

Figure 2.1: **Example Encoding Visualizations**

Example 3. *Continuing, the universe of features for this query log is (1)* ⟨ `_id`, `SELECT` ⟩, *(2)* ⟨ `_time`, `SELECT` ⟩, *(3)* ⟨ `sms_type`, `SELECT` ⟩, *(4)* ⟨ `status = ?`, `WHERE` ⟩, *(5)* ⟨ `sms_type = ?`, `WHERE` ⟩, *and (6)* ⟨ `Messages`, `FROM` ⟩. *Accordingly, the queries can be encoded as feature vectors, with fields recording each feature's presence:* $\mathbf{q}_1 = \langle\, 1,0,0,1,0,1 \,\rangle$, $\mathbf{q}_2 = \langle\, 0,1,0,1,1,1 \,\rangle$, $\mathbf{q}_3 = \langle\, 1,0,0,1,0,1 \,\rangle$, $\mathbf{q}_4 = \langle\, 0,1,1,0,1,1 \,\rangle$

Patterns. Our target applications require us to count the number of times features (co-)occur in a query. For example, materialized view selection requires counting tables used together in queries. Motivated by this observation, we begin by defining a broad class of *pattern-based encodings* that directly encode co-occurrence probabilities. A *pattern* is an arbitrary set of features $\omega = (x_1, \ldots, x_n)$ that may co-occur together. Each pattern captures a piece of information from the distribution $p(Q \mid L)$. In particular, we are interested in the probability of uniformly drawing a query $\mathbf{q}$ from the log that *contains* the pattern $\mathbf{b}$ (i.e., $\mathbf{q} \supseteq \omega$):

$$p(Q \supseteq \omega \mid L) = \textstyle\sum_{\mathbf{q} \in L \wedge \mathbf{q} \supseteq \omega} p(\mathbf{q} \mid L)$$

When it is clear from context, we abuse notation and write $p(\cdot)$ instead of $p(\cdot \mid L)$. Recall that $p(Q)$ can be represented as a joint distribution of variables $(X_1, \ldots, X_n)$ and probability $p(Q \supseteq \omega)$ is equivalent to $p(X_1 \geq x_1, \ldots, X_n \geq x_n)$.

Pattern-Based Encodings. Denote by $\mathcal{E}_{max} : \{0,1\}^n \to [0,1]$, the mapping from each pattern (ω) to its frequency in the log: $\mathcal{E}_{max} = \left\{ \left(\omega \to p(\omega) \right) \mid \omega \in \{0,1\}^n \right\}$

A *pattern-based encoding* $\mathcal{E}$ is any such partial mapping $\mathcal{E} \subseteq \mathcal{E}_{max}$. We denote the frequency of pattern ω in encoding $\mathcal{E}$ by $\mathcal{E}[\omega]$ ($= p(Q \supseteq \omega)$). When it is clear from context, we abuse syntax and also use $\mathcal{E}$ to denote the set of patterns it maps (i.e., *domain*($\mathcal{E}$)). Hence, $|\mathcal{E}|$ is the number of mapped patterns, which we call the encoding's *Verbosity*. A *pattern-based encoder* is any algorithm $\texttt{encode}(L, \epsilon) \mapsto \mathcal{E}$ whose input is a log L and whose output is a set of patterns $\mathcal{E}$, with Verbosity thresholded at some integer ϵ. Many pattern mining algorithms [15, 16] can be used for this purpose.

2.2.3.2 Communicating Information Content

A side-benefit of pattern-based encodings is that, under the assumption of isomorphism in Section 2.2.1, patterns can be translated to their query representations and used for human inspection of the log. Figure 2.1 shows two examples. The approach illustrated in Figure 2.1a uses shading to show each feature's frequency in the log, and communicates frequently occurring predicates or columns. This approach might, for example, help a human to manually select indexes. A second approach illustrated in Figure 2.1b conveys correlations, showing the frequency of entire patterns. The accompanying technical report [20] explores visualizations of pattern-based summaries in greater depth.

2.3 Information Loss

Our goal is to encode the distribution $p(Q)$ as a set of patterns: obtaining a less verbose encoding (i.e., with fewer patterns), while also ensuring that the encoding

captures $p(Q)$ with minimal information loss. In this section, we define information loss for pattern-based encodings.

2.3.1 Lossless Summaries

To establish a baseline for measuring information loss, we begin with the extreme cases. At one extreme, an empty encoding ($|\mathcal{E}| = 0$) conveys no information. At the other extreme, we have the encoding $\mathcal{E}_{max}$ which is the full mapping from all patterns. Having this encoding is a sufficient condition to exactly reconstruct the original distribution $p(Q)$.

Proposition 1. *For any query* $\mathbf{q} = (x_1, \ldots, x_n) \in 0, 1^n$*, the probability of drawing exactly* $\mathbf{q}$ *at random from the log (i.e.,* $p(X_1 = x_1, \ldots, X_n = x_n)$*) is computable, given* $\mathcal{E}_{max}$*.*

2.3.2 Lossy Summaries

Although $\mathcal{E}_{max}$ is lossless, its Verbosity is exponential in the number of features (n). Hence, we will focus on lossy encodings that can be less verbose. A lossy encoding $\mathcal{E} \subset \mathcal{E}_{max}$ may not precisely identify the distribution $p(Q)$, but can still be used to approximate it. We characterize the information content of a lossy encoding $\mathcal{E}$ by defining a *space* (denoted by $\Omega_{\mathcal{E}}$) of distributions $\rho \in \Omega_{\mathcal{E}}$ allowed by an encoding $\mathcal{E}$. This space is defined by constraints as follows: First, we have the general properties of probability distributions:

$$\forall \mathbf{q} \in \{0, 1\}^n : \rho(\mathbf{q}) \geq 0 \qquad \sum_{\mathbf{q}} \rho(\mathbf{q}) = 1$$

Each pattern $\mathbf{b}$ in the encoding $\mathcal{E}$ constrains relevant probabilities in distribution ρ to sum to the target frequency:

$$\forall \mathbf{b} \in \mathit{domain}(\mathcal{E}) : \quad \mathcal{E}[\mathbf{b}] = \sum\nolimits_{\mathbf{q} \supseteq \mathbf{b}} \rho(\mathbf{q})$$

Note that the dual constraints $1 - \mathcal{E}[\mathbf{b}] = \sum_{\mathbf{q} \not\supseteq \mathbf{b}} \rho(\mathbf{q})$ are redundant under constraint $\sum_{\mathbf{q}} \rho(\mathbf{q}) = 1$.

The resulting space $\Omega_{\mathcal{E}}$ is the set of all query logs, or equivalently the set of all possible distributions of queries, that obey these constraints. From the outside observer's perspective, the distribution $\rho \in \Omega_{\mathcal{E}}$ that the encoding conveys is ambiguous: We model this ambiguity using a random variable $\mathcal{P}_{\mathcal{E}}$ with support $\Omega_{\mathcal{E}}$. The true distribution $p(Q)$ derived from the query log must appear in $\Omega_{\mathcal{E}}$, denoted as $p(Q) \equiv \rho^* \in \Omega_{\mathcal{E}}$ (i.e., $p(\mathcal{P}_{\mathcal{E}} = \rho^*) > 0$). Of the remaining distributions ρ admitted by $\Omega_{\mathcal{E}}$, it is possible that some are more likely than others. For example, a query containing a column (e.g., `status`) is only valid if it also references a table that contains the column (e.g., `Messages`). This prior knowledge may be modeled as a prior on the distribution of $\mathcal{P}_{\mathcal{E}}$ or equivalently by an additional constraint. However, for the purposes of this paper, we take the uninformed prior by assuming that $\mathcal{P}_{\mathcal{E}}$ is uniformly distributed over $\Omega_{\mathcal{E}}$:

$$p(\mathcal{P}_{\mathcal{E}} = \rho) = \begin{cases} \frac{1}{|\Omega_{\mathcal{E}}|} & \text{if } \rho \in \Omega_{\mathcal{E}} \\ 0 & \text{otherwise} \end{cases}$$

Naive Encodings. One specific family of lossy encodings that treat each feature as being independent (e.g., as in Figure 2.1a) is of particular interest to us. We call this family *naive encodings*, and return to it throughout the rest of the paper. A naive

encoding $\ddot{\mathcal{E}}$ is composed of all patterns that have exactly one feature with non-zero frequency.

$$domain(\ddot{\mathcal{E}}) = \{\ (0, \ldots, 0, x_i, 0, \ldots, 0) \mid\ i \in [1, n],\ x_i = 1\ \}$$

2.3.3 Idealized Information Loss Measures

Based on the space of distributions constrained by the encoding, the information loss of an encoding can be considered from two related, but subtly distinct perspectives: (1) *Ambiguity* measures how much room the encoding leaves for interpretation and (2) *Deviation* measures how reliably the encoding approximates the target distribution $p(Q)$.

Ambiguity. We define the Ambiguity $\mathrm{I}(\mathcal{E})$ of an encoding as the entropy of the random variable $\mathcal{P}_{\mathcal{E}}$. The higher the entropy, the less precisely $\mathcal{E}$ identifies a specific distribution.

$$\mathrm{I}(\mathcal{E}) = \sum_{\rho} p(\mathcal{P}_{\mathcal{E}} = \rho) \log\left(p(\mathcal{P}_{\mathcal{E}} = \rho)\right)$$

Deviation. The deviation from any permitted distribution ρ to the true distribution ρ^* can be measured by the Kullback-Leibler (K-L) divergence $\mathcal{D}_{KL}(\rho^* || \rho)$. We define the Deviation $\mathrm{d}(\mathcal{E})$ of a encoding as the expectation of the K-L divergence over all permitted $\rho \in \Omega_{\mathcal{E}}$:

$$\mathrm{d}(\mathcal{E}) = \mathbb{E}_{\mathcal{P}_{\mathcal{E}}}\left[\mathcal{D}_{KL}(\rho^* || \mathcal{P}_{\mathcal{E}})\right] = \sum_{\rho \in \Omega_{\mathcal{E}}} p(\mathcal{P}_{\mathcal{E}} = \rho) \cdot \mathcal{D}_{KL}(\rho^* || \rho)$$

Limitations. There are two limitations to these idealized measures in practice. First, K-L divergence is not defined on any permitted distribution ρ where the true distribution ρ^* is not *absolutely continuous* (denoted $\rho^* \ll \rho$). Second, neither

Deviation nor Ambiguity has a closed-form formula.

2.4 Practical Loss Measure

Computing either Ambiguity or Deviation requires enumerating the entire space of permitted distributions. One approach to approximating either measure is repeatedly sampling from, rather than enumerating the space. However, accurate approximations require a large number of samples, rendering this approach similarly inefficient. In this section, we propose a faster approach to assessing the fidelity of a pattern encoding. Specifically, we select a single representative distribution $\overline{\rho}_{\mathcal{E}}$ from the space $\Omega_{\mathcal{E}}$, and use $\overline{\rho}_{\mathcal{E}}$ to approximate both Ambiguity and Deviation.

2.4.1 Reproduction Error

Maximum Entropy Distribution. The representative distribution is chosen by applying the maximum entropy principle [21] commonly used in pattern-based summarization [15, 16]. That is, we select the distribution $\overline{\rho}_{\mathcal{E}}$ with maximum entropy:

$$\overline{\rho}_{\mathcal{E}} = \arg\max_{\rho \in \Omega_{\mathcal{E}}} \mathcal{H}(\rho) \qquad \text{where } \mathcal{H}(\rho) = \sum_{\mathbf{q} \in \{0,1\}^n} -\rho(\mathbf{q}) \log \rho(\mathbf{q})$$

The maximum entropy distribution best represents the current state of knowledge. That is, a distribution with lower entropy assumes additional constraints derived from patterns that we do not know, while one with higher entropy violates the constraints from patterns we do know.

Maximizing an objective function belonging to the exponential family (entropy in our case) under a mixture of linear equalities/inequality constraints is a convex optimization problem [22] which guarantees a *unique* solution and can be efficiently

solved using the cvx toolkit [23, 24], and/or by *iterative scaling* [15, 16]. For naive encodings specifically, we can assume independence between each feature X_i. Under this assumption, $\overline{\rho}_{\mathcal{E}}$ has a closed-form representation:

$$\overline{\rho}_{\mathcal{E}}(\mathbf{q}) = \prod_i p(X_i = x_i) \qquad \text{where } \mathbf{q} = (x_1, \ldots, x_n)$$

We define *Reproduction Error* $e(\mathcal{E})$ as the entropy difference between the representative and true distributions:

$$e(\mathcal{E}) = \mathcal{H}(\overline{\rho}_{\mathcal{E}}) - \mathcal{H}(\rho^*) \qquad \text{where } \overline{\rho}_{\mathcal{E}} = \arg\min_{\rho \in \Omega_{\mathcal{E}}} -\mathcal{H}(\rho)$$

2.4.2 Practical vs Idealized Information Loss

In this section we prove that Reproduction Error closely parallels Ambiguity. We define a partial order lattice over encodings and show that for any pair of encodings on which the partial order is defined, a like relationship is implied for both Reproduction Error and Ambiguity. We supplement the proofs given in this section with an empirical analysis relating Reproduction Error to Deviation in Section 2.7.1.

Containment. We define a partial order over encodings $\leq_\Omega$ based on *containment* of their induced spaces $\Omega_{\mathcal{E}}$:

$$\mathcal{E}_1 \leq_\Omega \mathcal{E}_2 \quad \equiv \quad \Omega_{\mathcal{E}_1} \subseteq \Omega_{\mathcal{E}_2}$$

That is, one encoding (i.e., $\mathcal{E}_1$) precedes another (i.e., $\mathcal{E}_2$) when all distributions admitted by the former encoding are also admitted by the latter.

Containment Captures Reproduction Error. We first prove that the total order given by Reproduction Error is a superset of the partial order $\leq_\Omega$.

Lemma 1. *For any pair of encodings $\mathcal{E}_1, \mathcal{E}_2$ that induce spaces $\Omega_{\mathcal{E}_1}, \Omega_{\mathcal{E}_2}$ and maximum entropy distributions $\overline{\rho}_{\mathcal{E}_1}, \overline{\rho}_{\mathcal{E}_2}$ it holds that $\mathcal{E}_1 \leq_\Omega \mathcal{E}_2 \rightarrow e(\mathcal{E}_1) \leq e(\mathcal{E}_2)$.*

Proof. First we have $\Omega_{\mathcal{E}_2} \supseteq \Omega_{\mathcal{E}_1} \rightarrow \overline{\rho}_{\mathcal{E}_1} \in \Omega_{\mathcal{E}_2}$. Since $\overline{\rho}_{\mathcal{E}_2}$ has the maximum entropy among all distributions $\rho \in \Omega_{\mathcal{E}_2}$, we have $\mathcal{H}(\overline{\rho}_{\mathcal{E}_1}) \leq \mathcal{H}(\overline{\rho}_{\mathcal{E}_2}) \equiv e(\mathcal{E}_1) \leq e(\mathcal{E}_2)$. □

Containment Captures Ambiguity. Next, we show that the partial order based on containment implies a like relationship between Ambiguities of pairs of encodings.

Lemma 2. *Given encodings $\mathcal{E}_1, \mathcal{E}_2$ with uninformed prior on $\mathcal{P}_{\mathcal{E}_1}, \mathcal{P}_{\mathcal{E}_2}$, it holds that $\mathcal{E}_1 \leq_\Omega \mathcal{E}_2 \rightarrow I(\mathcal{E}_1) \leq I(\mathcal{E}_2)$.*

Proof. Given an uninformed prior: $\mathrm{I}(\mathcal{E}) = \log|\Omega_\mathcal{E}|$, we have $\mathcal{E}_1 \leq_\Omega \mathcal{E}_2 \rightarrow |\Omega_{\mathcal{E}_1}| \leq |\Omega_{\mathcal{E}_2}| \rightarrow \mathrm{I}(\mathcal{E}_1) \leq \mathrm{I}(\mathcal{E}_2)$ □

2.5 Pattern Mixture Encodings

Thus far we have defined the problem of log compression, treating the query log as a multivariate distribution $p(Q)$ where patterns capture positive frequencies of feature (co-)occurrence. However in cases like logs of *mixed* workloads, there are also many cases of anti-correlation between features. For example, consider a log that includes queries drawn from a mixture of two workloads with disjoint feature sets. Pattern-based summaries can not convey such anti-correlations easily. As a result, patterns including features from both workloads never actually co-occur in the log, but a pattern-based summary of the log will suggest otherwise. Such false positives are especially problematic for use-cases of LogR involving outlier detection (e.g., [25]). Even in other settings, capturing correlations reduces data dimensionality and improves both runtime and effectiveness of state-of-the-art pattern mining algorithms (See Section 2.8.1).

In this section, we propose a generalization of pattern encodings where the log is modeled not as a single probability distribution, but rather as a mixture of several simpler distributions. The resulting encoding is likewise a mixture: Patterns for each component of the mixture are stored independently. Hence, we refer to it as a *pattern mixture encoding*, and it forms the basis of LogR compression. We first focus on a simplified form of this problem, where we only mix *naive* encodings (we explore more general mixtures in Section 2.6.4). We refer to the resulting scheme as *naive mixture encodings*, and give examples of the encoding in Section 2.5.1. Then we generalize Reproduction Error and Verbosity to pattern mixture encodings in Section 2.5.2. Finally, with generalized encoding evaluation measures, we evaluate several clustering methods for creating naive mixture encodings.

2.5.1 Example: Naive Mixture Encodings

Consider a toy query log with only 3 conjunctive queries.

1. `SELECT id FROM Messages WHERE status = ?`
2. `SELECT id FROM Messages`
3. `SELECT sms_type FROM Messages`

The codebook of this log includes 4 features: $\langle$ `id`, `SELECT` $\rangle$, $\langle$ `sms_type`, `SELECT` $\rangle$, $\langle$ `Messages`, `FROM` $\rangle$, $\langle$ `status = ?`, `WHERE` $\rangle$. Re-encoding the three queries as vectors, we get:

$$1.\langle 1,0,1,1 \rangle \qquad 2.\langle 1,0,1,0 \rangle \qquad 3.\langle 0,1,1,0 \rangle$$

A naive encoding of this log can be expressed as:

$$\left\langle \frac{2}{3},\ \frac{1}{3},\ 1,\ \frac{1}{3} \right\rangle$$

This encoding captures that all queries in the log pertain to the `Messages` table, but obscures the relationship between the remaining features. For example, this encoding obscures the anti-correlation between `id` and `sms_type`. Similarly, the encoding hides the correlation between `status = ?` and `id`. Such relationships are critical for evaluating the effectiveness of views or indexes.

Example 4. *The maximum entropy distribution for any naive encoding assumes that features are independent. Assuming independence, the probability of query 1 uniformly drawn from the log is estimated as:*

$$p(\texttt{id}) \cdot p(\neg \texttt{sms\_type}) \cdot p(\texttt{Messages}) \cdot p(\texttt{status=?}) = \frac{4}{27} \approx 0.148$$

This is a significant difference from the true probability of this query (i.e., $\frac{1}{3}$). Conversely queries not in the log, such as the following, have non-zero probability in the encoding.

```
SELECT sms_type FROM Messages WHERE status = ?
```

$$p(\neg \texttt{id}) \cdot p(\texttt{sms\_type}) \cdot p(\texttt{Messages}) \cdot p(\texttt{status=?}) = \frac{1}{27} \approx 0.037$$

To achieve a more faithful representation of the original log, we could partition it into two components, with the corresponding encoding parameters:

Partition 1 (L_1)		**Partition 2** (L_2)
$(1,0,1,1)$	$(1,0,1,0)$	$(0,1,1,0)$
$\downarrow$	$\downarrow$	$\downarrow$
$\langle\ 1,\ 0,\ 1,\ \frac{1}{2}\ \rangle$		$\langle\ 0,\ 1,\ 1,\ 0\ \rangle$

Although there are now two encodings, the encodings are not ambiguous. The feature `status = ?` appears in exactly half of the log entries, and is indeed independent of the other features. All other attributes in each encoding appear in all queries in their respective partitions. Furthermore, the maximum entropy distribution induced by each encoding is exactly the distribution of queries in the partitioned log. Hence, the Reproduction Error is zero for both encodings.

2.5.2 Generalized Encoding Fidelity

We next generalize our definitions of Reproduction Error and Verbosity from pattern-based to pattern mixture encodings. Suppose query log L has been partitioned into K clusters with L_i, $\mathcal{E}_i$, $\overline{\rho}_{\mathcal{E}_i}$ and ρ_i^* (where $i \in [1, K]$) representing the log of queries, encoding, maximum entropy distribution, and true distribution (respectively) for the ith cluster. First, observe that the distribution for the whole log (i.e., ρ^*) is the sum of distributions for each partition (i.e., ρ_i^*) weighted by the proportion (i.e., $\frac{|L_i|}{|L|}$) of queries:

$$\rho^*(\mathbf{q}) = \sum_{i=1,\ldots,K} w_i \cdot \rho_i^*(\mathbf{q}) \qquad \text{where } w_i = \frac{|L_i|}{|L|}$$

Generalized Reproduction Error. Similarly, the maximum entropy distribution $\overline{\rho}_{\mathcal{E}}$ for the whole log is:

$$\overline{\rho}_{\mathcal{E}}(\mathbf{q}) = \sum_{i=1,\ldots,K} w_i \cdot \overline{\rho}_{\mathcal{E}_i}(\mathbf{q})$$

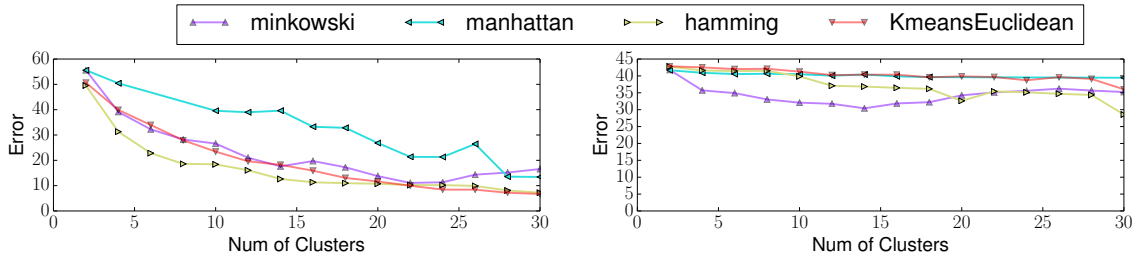
We define the *Generalized Reproduction Error* of a pattern mixture encoding similarly, as the weighted sum of Reproduction Error for each partition:

$$e(\mathcal{E}) = \mathcal{H}(\overline{\rho}_{\mathcal{E}}) - \mathcal{H}(\rho^*) = \sum_i w_i(\mathcal{H}(\overline{\rho}_{\mathcal{E}_i}) - \mathcal{H}(\rho_i^*)) = \sum_i w_i e(\mathcal{E}_i)$$

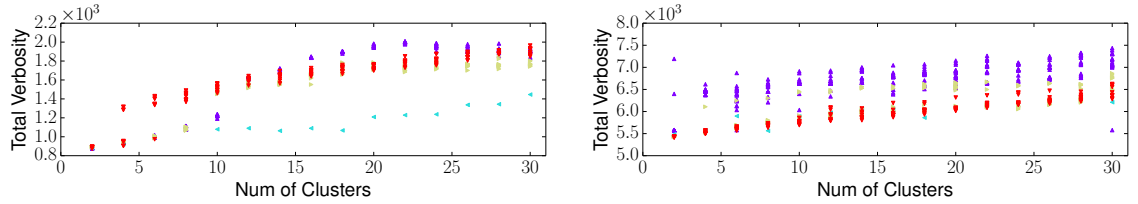
When it is clear from context, we refer to Generalized Reproduction Error as Error

in the rest of this paper. As in the base case, a pattern mixture encoding with low Error indicates a high-fidelity representation of the original log. A process can infer the frequency of any query $p(Q = \mathbf{q} \mid L)$ drawn from the original distribution, simply by inferring its frequency in each cluster i (i.e., $p(Q = \mathbf{q} \mid L_i)$) and taking a weighted average over all inferences.

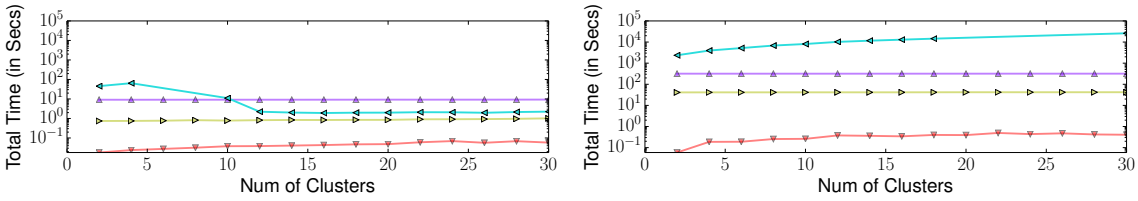
Generalized Verbosity. We generalize verbosity to pattern mixture encodings as the *Total Verbosity* ($\sum_i |S_i|$), or the total size of the encoded representation.



(a) **Error v. Number of Clusters (Left: PocketData, Right: US Bank)**



(b) **Total Verbosity v. Number of Clusters (Left: PocketData, Right: US Bank); Each point is the verbosity of one of ten trials. The Y-axis' lower bound is the verbosity at 1 cluster to better show the change in verbosity.**



(c) **Runtime v. Number of Clusters (Left: PocketData, Right: US Bank)**

Figure 2.2: **Clustering Schemes Comparison**

2.6 Pattern Mixture Compression

We are now ready to describe the LogR compression scheme. Broadly, LogR attempts to identify a pattern mixture encoding that optimizes for some target trade-off between Total Verbosity and Error. A naive — though impractical — approach to finding such an encoding would be to search the entire space of possible pattern mixture encodings. Instead, LogR approximates the same outcome by first identifying the naive mixture encoding that is closest to optimal for the desired trade-off. As we show experimentally, this naive mixture encoding is competitive with more complicated, slower techniques for summarizing query logs. We also explore a hypothetical second stage, where LogR refines the naive mixture encoding to further reduce Error. The outcome of this hypothetical stage has a slightly lower Error and Verbosity, but does not admit efficient computation of database statistics.

2.6.1 Constructing Naive Mixture Encodings

LogR searches for a naive mixture encoding that best optimizes for a requested tradeoff between Total Verbosity and Error. As a way to make this search efficient, we observe that a log (or log partition) uniquely determines its naive (or naive mixture) encoding. Thus the problem of searching for a naive mixture encoding reduces to searching for the corresponding log partitioning. We further observe that the Error of a naive mixture encoding is proportional to the diversity of the queries in the log being encoded: The more uniform the log (or partition), the lower the Error. Hence, the partitioning problem further reduces to clustering queries in the log by feature overlap. To identify a suitable clustering scheme, we next evaluate four commonly used clustering schemes with respect to their ability to create naive mixture encodings with low Error and Verbosity: (1) KMeans [26] with Euclidean

distance (i.e., l_2-norm) and Spectral Clustering [27] with (2) Manhattan (i.e., l_1-norm), (3) Minkowski (i.e., l_p-norm) with $p = 4$, and (4) Hamming distances[1].

Experiment Setup. Spectral and KMeans clustering algorithms are implemented by *sklearn* [28] in Python. We gradually increase K (i.e., the number of clusters) for each clustering scheme to mimic the process of continuously sub-clustering the log, tolerating higher Total Verbosity for lower Error. To reduce randomness in clustering, we run each of them 10 times for each K and averaging the Error of the resulting encodings. We used two datasets: "US Bank" and "PocketData". We describe both datsets and the data preparation process in detail in Section 2.7. All results for our clustering experiments are shown in Figure 2.2.

2.6.1.1 Clustering

We next show that clustering is an effective way to consistently reduce Error, although no one clustering scheme is ideal for all three of Error, Verbosity, and runtime.

More clusters reduces Error. Figure 2.2a compares the relationship between the number of clusters (x-axis) and Error (y-axis), showing the varying rates of convergence to zero Error for each clustering scheme. We observe that adding more clusters does consistently reduce Error for both data sets, regardless of clustering algorithm or distance measure. We note that the US Bank dataset is significantly more diverse than the PocketData dataset, with respect to the total number of features (See Table 2.1) and that more than 30 clusters may be required for reaching near-zero Error. In general, Hamming distance converges faster than other distance measures on PocketData. Minkowski distance shows faster convergence rate than

[1]We also evaluated Spectral Clustering with Euclidean, Chebyshev and Canberra distances; These did not perform better and we omit them in the interest of conciseness.

Hamming within 14 clusters on the US bank dataset.

Adding more clusters increases Verbosity. Figure 2.2b compares the relationship between the number of clusters (x-axis) and Verbosity (y-axis). We observe that Verbosity increases with the number of clusters. This is because when a partition is split, each feature common to both partitions increases the Verbosity by one.

Hierarchical Clustering. The clustering schemes produce non-monotonic cluster assignments. That is, Error can occasionally grow as clusters are added (Figure 2.2a). An alternative is to use hierarchical clustering [26], which forces monotonic assignments and offers more dynamic control over the Error/Verbosity tradeoff.

Run Time Comparison. The total runtime (y-axis) in Figure 2.2c includes both distance matrix computation time (if any) and clustering time. Note the log-scale: K-Means is orders of magnitude faster than the others.

Take-Aways. For time-sensitive applications, KMeans algorithm is preferred to Spectral Clustering. With respect to distance measures, minkowski (i.e., l_p-norm) with $p = 4$ provides the best tradeoff between Error and runtime.

Visualizing Naive Mixture Encoding. As with normal pattern summaries, naive mixture summaries are also interpretable. For example a visualization like that of Figure 2.1a can be repeated, once for each cluster. For more details, see our accompanying technical report [20].

2.6.2 Approximating Log Statistics

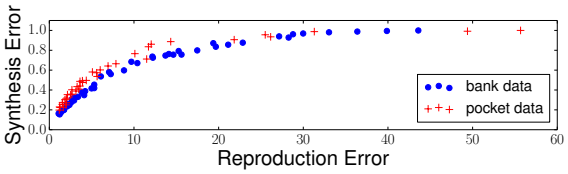
Recall that our primary goal is estimating statistical properties. In particular, we are interested in counting the occurrences $\Gamma_\omega(L)$ (i.e., $p(Q \supseteq \mathbf{b}) \cdot |L|$) of some pattern ω in the log. Recall that a naive encoding $\ddot{\mathcal{E}}$ includes only single-feature patterns (i.e., patterns exactly encoding $p(X_i \geq x_i)$) and that the closed-form representation

for the maximum entropy distribution $\overline{\rho}_{\ddot{\mathcal{E}}}$ arises by independence between features (i.e., $\overline{\rho}_{\ddot{\mathcal{E}}}(Q = \mathbf{q}) = \prod_i p(X_i = x_i)$). Similarly, we use the independence assumption to estimate:

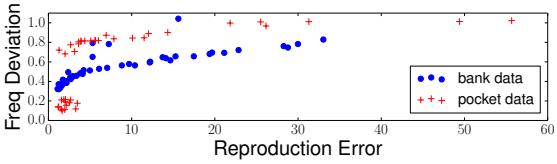
$$est[\Gamma_\omega(L) \mid \ddot{\mathcal{E}}] = \overline{\rho}_{\ddot{\mathcal{E}}}(Q \supseteq \mathbf{b}) \cdot |L| = \prod_i p(X_i \geq x_i) \cdot |L|$$

This process trivially generalizes to naive pattern mixture encodings by mixing distributions. Specifically, given a set of partitions $L_1 \cup \ldots \cup L_K = L$, the estimated counts for $\Gamma_\omega(L)$ under each individual partition L_i can be computed based on the partition's naive encoding $\ddot{\mathcal{E}}_i$, and we then sum up the estimated counts in each partition:

$$est[\ \Gamma_\omega(L_i) \mid \ddot{\mathcal{E}}_1, \ldots, \ddot{\mathcal{E}}_K\ ] = \sum_{i \in [1,K]} est[\ \Gamma_\omega(L_i) \mid \ddot{\mathcal{E}}_i\ ]$$



(a) **Synthesis Error v. Reproduction Error**



(b) **Frequency Deviation v. Reproduction Error**

Figure 2.3: **Effectiveness of Naive Mixture Encoding**

2.6.3 Pattern Synthesis & Frequency Estimate

In this section, we empirically verify the effectiveness of naive mixture encodings in approximating log statistics from two related perspectives. The first perspective

focuses on *synthesis error*. It measures whether patterns synthesized by the naive mixture encoding actually appear in the log. From the second perspective, we further investigate the *frequency deviation* of patterns contained in the log. This evaluates whether a naive mixture encoding computes the correct frequency for patterns of interest to client applications. Experimental results are shown in Figure 2.3. Both synthesis error and frequency deviation consistently decrease given more clusters. Furthermore, as we vary the number of clusters, both measures correlate with Reproduction Error.

Synthesis Error is measured by $1 - \frac{m}{n}$ where m out of n randomly synthesized patterns actually appear in the log. Intuitively, when synthesis error grows, it is more likely that a pattern from the synthesized log will not appear in the original log (i.e., smaller values are better). Figure 2.3a shows synthesis error (y-axis) versus Reproduction Error (x-axis). The figure is generated by synthesizing $n = 10000$ patterns from each cluster of the log. Note that different values of n give similar observations. The overall synthesis error is measured by the average of synthesis errors for all clusters, weighted by the proportion of queries in each cluster.

Frequency Deviation is measured for a pattern by $\frac{|est-t|}{t}$ where t stands for true frequency of a pattern and *est* is the one estimated by the naive mixture encoding. Since frequency deviation is smaller when evaluated on a pattern contained in the other, as an alternative, we treat each distinct query in the log as a pattern and the frequency deviation on it will be the worst case for all patterns that it contains. Intuitively, this value captures the percentage error of frequency estimates (i.e., smaller values are better). For each cluster, we sum frequency deviations on all of its distinct queries and the final frequency deviation for the whole log is an weighted average (same as synthesis error) over all clusters. Figure 2.3b shows frequency deviation (y-axis) versus Reproduction Error (x-axis).

2.6.4 Naive Encoding Refinement

Naive mixture encodings can already achieve close to near-zero Error (Figure 2.2a), have low Total Verbosity, and admit efficiently computable log statistics $\Gamma_\omega(L)$. Doing so makes estimating statistics more computationally expensive. However, as a thought experiment we consider a hypothetical second pass to enrich naive mixture encodings with non-naive patterns. We start by considering the simpler problem of identifying the *individual* non-naive pattern that maximally reduces the Reproduction Error of a naive encoding.

Feature-Correlation Refinement. Recall that under naive encodings, we have a closed-form estimation $\overline{\rho}_{\ddot{\mathcal{E}}}(Q \supseteq \mathbf{b})$ of pattern frequencies $p(Q \supseteq \mathbf{b})$. We thus define the *feature-correlation* of pattern $\mathbf{b}$ as the log-difference from its actual frequency to the estimate.

$$fc(\mathbf{b}, \ddot{\mathcal{E}}) = |\log(p(Q \supseteq \mathbf{b})) - \log(\overline{\rho}_{\ddot{\mathcal{E}}}(Q \supseteq \mathbf{b}))|$$

Intuitively, patterns with higher feature correlations carry more information content of the log that its naive encoding ignores, making them ideal candidates for addition to the naive encoding. For two patterns with the same feature-correlation, the one that occurs more frequently [29] will have greater impact on Reproduction Error. As a result, we compute an overall score for ranking individual patterns:

$$corr\_rank(\mathbf{b}) = p(Q \supseteq \mathbf{b}) \cdot fc(\mathbf{b}, \ddot{\mathcal{E}})$$

We show in Section 2.7.1 that *corr_rank* closely correlates with Reproduction Error. That is, a higher *corr_rank* value indicates that a pattern produces a greater reduction in Reproduction Error if introduced into the naive encoding.

Pattern Diversification. In general, we would like to identify a *set* of patterns.

The greedy approach that adds patterns one by one based on their ranking scores *corr_rank* is unreliable, as modifying the naive encoding invalidates the closed-form estimation $\overline{\rho}_{\tilde{\mathcal{E}}}(Q \supseteq \mathbf{b})$ that score *corr_rank* relies on. In other words, we can not sum up *corr_rank* scores of patterns in a set to rank its overall contribution to Reproduction Error reduction, as information content carried by patterns may overlap. To counter such overlap, or equivalently to *diversify* patterns, a search through the space of pattern-sets is needed. This type of diversification is commonly used in pattern mining applications, but can quickly become expensive. As we show experimentally in Section 2.7.2, the benefit of diversification is minimal.

2.7 Experiments

In this section, we design experiments to empirically (1) validate that Reproduction Error correlates with Deviation and (2) evaluate the effectiveness of LogR compression.

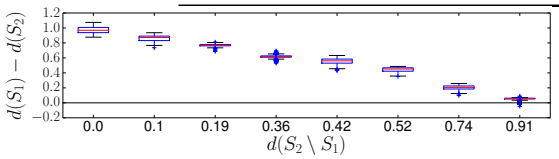
We use two specific datasets in the experiment: (1) SQL query logs of the Google+ Android app extracted from the PocketData public dataset [30] and (2) SQL query logs that capture all query activity on the majority of databases at a major US bank over a period of approximately 19 hours. A summary of these two datasets is given in Table 2.1.

The PocketData-Google+ query log. The dataset consists of SQL logs that capture all database activities of 11 Android phones. We selected the Google+ application for our study since it is one of the few applications where all users created a workload. This dataset is a stable workload of exclusively machine-generated queries.

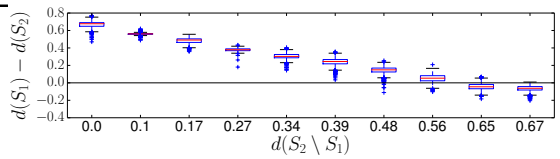
The US bank query log. This log is an anonymized record of queries processed by multiple relational database servers at a major US bank [6] over a period of 19 hours.

Table 2.1: **Summary of Data sets**

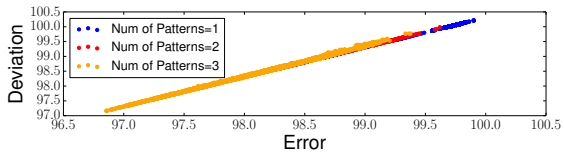
Statistics	PocketData	US bank
# Queries	629582	1244243
# Distinct queries	605	188184
# Distinct queries (w/o const)	605	1712
# Distinct conjunctive queries	135	1494
# Distinct re-writable queries	605	1712
Max query multiplicity	48651	208742
# Distinct features	863	144708
# Distinct features (w/o const)	863	5290
Average features per query	14.78	16.56



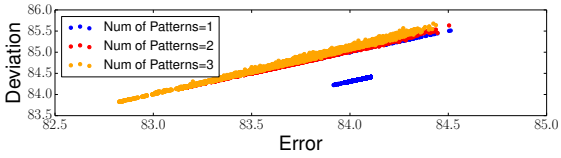
(a) **Containment captures Deviation (US bank)**



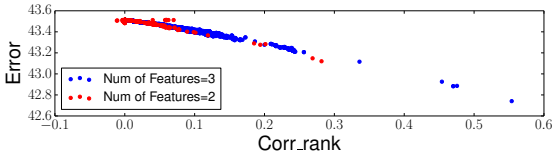
(b) **Containment captures Deviation (PocketData)**



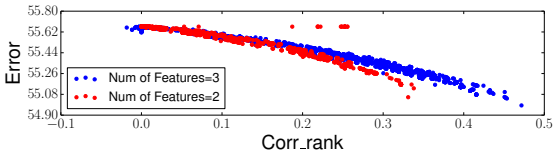
(c) **Error captures Deviation (US bank)**



(d) **Error captures Deviation (PocketData)**



(e) **Error captures Correlation (US bank)**



(f) **Error captures Correlation (PocketData)**

Figure 2.4: **Validating Reproduction Error**

Of the nearly 73 million database operations captured, 58 million are not directly queries, but rather invocations of stored procedures. A further 13 million used non-standard SQL features not supported by our SQL parser. Of the remaining of the 2.3 million parsed SQL queries, we base our analysis on the 1.25 million conjunctive

`SELECT` queries. This dataset can be characterized as a diverse workload of both machine- and human-generated queries.

Common Experiment Settings. Experiments were performed on a 2.8 GHz Intel Core i7 CPU with 16 GB 1600 MHz DDR3 memory and a SSD running macOS Sierra.

Constant Removal. A number of queries in the US bank query log differ only in hard-coded constant values. Table 2.1 shows the total number of queries, as well as the number of distinct queries if we ignore constants. By comparison, queries in PocketData all use JDBC parameters. For these experiments, we ignore constant values in queries.

Query Regularization. We apply query rewrite rules (same as [31]) to regularize queries into equivalent conjunctive forms, where possible. Table 2.1 shows that $\frac{135}{605}$ and $\frac{1494}{1712}$ of distinct queries are in conjunctive form for PocketData and US bank respectively. After regularization, all queries in both data sets can be either simplified into conjunctive queries or re-written into a `UNION` of conjunctive queries compatible with feature scheme of Aligon et al. [17].

Convex Optimization Solving. All convex optimization problems for measuring Reproduction Error and Deviation are solved by the *successive approximation heuristic* implemented by the CVX toolbox [23] with the Sedumi solver.

2.7.1 Validating Reproduction Error

In this section, we validate that Reproduction Error is a practical alternative to Deviation. In addition, we also offer measurements on its correlation with Deviation and score *corr_rank* in Section 2.6.4. As it is impractical to enumerate all possible encodings, we choose a subset of encodings for both datasets. Specifically, we first

select all features with frequencies in the range $[0.01, 0.99]$ and use these features to construct patterns. We then enumerate combinations of K (up to 3) patterns as our chosen encodings.

Containment Captures Deviation. Here we empirically verify that containment (Section 2.4.2) captures Deviation (i.e., $\mathcal{E}_1 \leq_\Omega \mathcal{E}_2 \rightarrow d(\mathcal{E}_1) \leq d(\mathcal{E}_2)$) to complete the chain of reasoning that Reproduction Error captures Deviation. Figures 2.4a and 2.4b show all pairs of encodings where $\mathcal{E}_2 \supset \mathcal{E}_1$. The y-axis shows the difference in Deviation values (i.e., $d(\mathcal{E}_2) - d(\mathcal{E}_1)$). Deviation $d(\mathcal{E})$ is approximated by drawing 1 million samples from the space $\Omega_\mathcal{E}$ induced by the encoding $\mathcal{E}$. For clarity, we bin pairs of encodings by the degree of overlap between them, measured by the Deviation of the set-difference $d(\mathcal{E}_2 \setminus \mathcal{E}_1)$; Higher $d(\mathcal{E}_2 \setminus \mathcal{E}_1)$ implies less overlap. Y-axis values are grouped into bins and visualized by boxplot where the boxes represent ranges within standard deviation and crosses are outliers. Intuitively, *points above zero* on the y-axis (i.e., $d(\mathcal{E}_2) - d(\mathcal{E}_1) > 0$) are pairs of encodings where the Deviation order agrees with containment order. This is the case for virtually all encoding pairs.

Additive Separability of Deviation. We also observe from Figures 2.4a and 2.4b that agreement between Deviation and containment order is correlated with overlap: More similar encodings are more likely to have agreement. Combined with Proposition 1, this shows first that for similar encodings, Reproduction Error is likely to be a reliable indicator of Deviation. This also suggests that Deviation is additively separable: The information loss (i.e., $d(\mathcal{E}_2) - d(\mathcal{E}_1)$) caused by excluding the encoding $\mathcal{E}_2 \setminus \mathcal{E}_1$ from $\mathcal{E}_2$ correlates with the quality (i.e., $d(\mathcal{E}_2 \setminus \mathcal{E}_1)$) of the encoding $\mathcal{E}_2 \setminus \mathcal{E}_1$ itself:

$$\mathcal{E}_2 \supset \mathcal{E}_1 \rightarrow d(\mathcal{E}_2) - d(\mathcal{E}_1) < 0 \quad \text{and} \quad d(\mathcal{E}_2 \setminus \mathcal{E}_1) \propto d(\mathcal{E}_2) - d(\mathcal{E}_1)$$

Error correlates with Deviation. As a supplement, Figures 2.4c and 2.4d empirically confirm that that Reproduction Error (x-axis) indeed closely correlates with Deviation (y-axis). Mirroring our findings above, correlation between them is tighter at lower Reproduction Error.

Error and Feature-Correlation. Figure 2.4e and 2.4f show the relationship between Reproduction Error (y-axis) and score *corr_rank* (x-axis), as discussed in Section 2.6.4. Values of y-axis are Reproduction Error of the naive encodings extended by a non-naive pattern $\mathbf{b}$ containing multiple features (up to 3 for illustrative purposes). One can observe that Reproduction Error of extended naive encodings almost linearly correlates with $corr\_rank(\mathbf{b})$. In addition, one can also observe that *corr_rank* becomes higher when the pattern $\mathbf{b}$ encodes more correlated features.

2.7.2 Feature-Correlation Refinement

In this section, we design experiments serving two purposes: (1) Evaluating the potential reduction of Error from refining naive mixture encodings through state-of-the-art pattern-based summarizers, and (2) Evaluating whether we can replace naive mixture encodings by the encodings created from summarizers that we have plugged-in.

Experiment Setup. To serve both purposes, we construct pattern mixture encodings under three configurations: (1) Naive mixture encodings; (2) Pattern-based encodings and (3) Naive mixture encodings refined into pattern-based encodings. Naive mixture encodings are constructed by K-Means clustering. Pattern-based encodings are generated by two state-of-the-art pattern-based summarizers: (1) *Laserlight* [16] that summarizes multi-dimensional data in order to predict an augmented binary variable and (2) *MTV* [15] that aims at mining maximally informative patterns that

summarize binary multi-dimensional data.

The experimental results are shown in Figure 2.5 that contains 3 sub-figures sharing the same x-axis, i.e., the number of clusters. Figure 2.5a compares the Error (y-axis) between naive mixture encodings and pattern mixture encodings that consist of patterns mined from *MTV* or *Laserlight*. Figure 2.5b evaluates the change in Error (y-axis) through refining naive mixture encodings by adding patterns from *MTV* or *Laserlight*. Figure 2.5c compares the runtime (y-axis) between constructing naive mixture encodings and applying *MTV* or *Laserlight*. We only show the results for US bank query log as results for PocketData give similar observations.

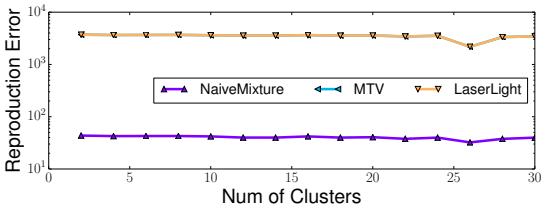
2.7.2.1 Pattern-based vs Naive Mixture Encodings

Figure 2.5a and 2.5c suggest that naive mixture encodings outperform pattern-based encodings in two ways.

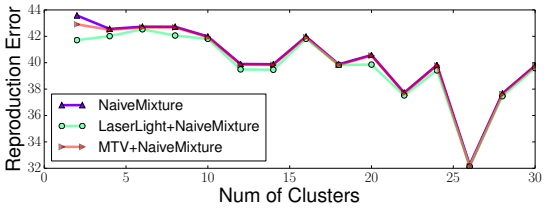
Reproduction Error. We observe from Figure 2.5a that the Reproduction Error of naive mixture encodings are orders of magnitude lower than pattern-based encodings generated by *Laserlight* or *MTV* alone.

Computation Efficiency. From Figure 2.5c we observe that the runtime of constructing naive mixture encodings is significantly lower than that of *Laserlight* and *MTV*.

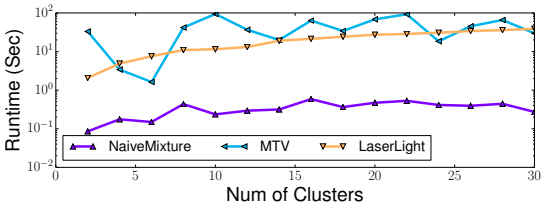
The one way where pattern-based encodings outperform naive mixture encodings is in Total Verbosity. *Laserlight* and *MTV* produce encodings with significantly fewer patterns, as the naive mixture encoding requires at least one pattern for each feature (e.g., 5290 patterns in the US bank query log). Conversely, mining this number of patterns is computationally infeasible (Figure 2.5c).



(a) **Naive Mixture v. LaserLight/MTV alone. Note that y-axis is in log scale.**



(b) **Naive Mixture v. Naive Mixture+LaserLight/MTV. Note that we offset y-axis (non-zero start).**



(c) **Runtime Comparison (y-axis in log scale)**

Figure 2.5: **Feature-correlation refinement (US bank)**

2.7.2.2 Refining Naive Mixture Encodings

The experiment result is shown in Figure 2.5b. Note that we offset y-axis to show the change in Error. We observe from the figure that reduction of Error contributed by plugging-in pattern-based summarizers is small for both algorithms.

Dimensionality Restriction. For *Laserlight*, the observation is partially due to the fact that we only keep top 100 features (in terms of variability) of the data as its input, since *Laserlight* is implemented in PostgresSQL 9.1 which has a threshold of 100 arguments (one argument for each feature) that can be passed to a function.

Pattern Restriction. For *MTV*, this is due to a runtime error that limits us to 15 or less patterns. We refer the reader to Section 4.5 in [15] that explains the difficulty in inferring the maximum entropy distribution constrained by a large number of non-naive patterns.

2.8 Alternative Applications

To fairly evaluate *Laserlight* and *MTV*, we incorporate their own data sets and empirically evaluate them against *naive mixture encoding* under their own applications.

Data Sets. Specifically, we choose *Mushroom* data set used in *MTV* [15] which is obtained from FIMI dataset repository and U.S. Census data on Income or simply *Income* data set, which is downloaded from IPUMS-USA at *https://usa.ipums.org/usa/* and used in *Laserlight* [16]. The basic statistics of the data sets are given in Table 2.2.

Table 2.2: **Data Sets of Alternative Applications**

Statistics	Income	Mushroom
# Distinct data tuples	777493	8124
# Features per tuple	9	21
Feature Binary-valued?	no	no
# Distinct features	783	95
Binary Classification Feature	$> 100,000$?	Edibility
Assumed data tuple multiplicity	1	1

2.8.1 Experiments

All experiments involving *Laserlight* and *MTV* will be evaluated under their own Error measures and data sets, unless otherwise stated. The experiments are organized as follows: First, we establish baselines by evaluating classical *Laserlight* and *MTV* on their original data; Then we show that classical *Laserlight* and *MTV* can be generalized to partitioned data and that the generalization improves on their Error measures and also runtime; At last, we compare their generalized versions with *naive mixture encoding* to show that *naive mixture encoding* is a reasonable alternative.

2.8.1.1 Error Measures

We first explain how *naive mixture encoding* is evaluated based on Error defined by *Laserlight* and *MTV*.

Evaluating Naive Encoding on Laserlight Error. Algorithm *Laserlight* summarizes data D which consists of feature vectors t augmented by some binary feature v. Denote the valuation of the binary feature v for each feature vector t as $v(t)$. The goal is to mine a summary encoding $\mathcal{E}$, which is a set of patterns contained in $t \in D$ that offer predictive power on $v(t)$. Denote the estimation (based on $\mathcal{E}$) of $v(t)$ as $u_{\mathcal{E}}(t) \in [0, 1]$, the *Laserlight* Error is measured by

$$\sum_t (v(t) \log(\frac{v(t)}{u_{\mathcal{E}}(t)}) + (1 - v(t)) \log(\frac{1 - v(t)}{1 - u_{\mathcal{E}}(t)}))$$

Since *naive encoding* $\ddot{\mathcal{E}}$ assumes feature independence, estimation of $v(t)$ is independent of t, namely $u_{\ddot{\mathcal{E}}}(t) = u_{\ddot{\mathcal{E}}} = |\{\tau | v(\tau) = 1, \tau \in D\}|/|D|$. Consequently, the *Laserlight* Error of *naive encoding* is

$$-|D|(u_{\ddot{\mathcal{E}}} \log u_{\ddot{\mathcal{E}}} + (1 - u_{\ddot{\mathcal{E}}}) \log(1 - u_{\ddot{\mathcal{E}}}))$$

Evaluating Naive Encoding on MTV Error. Given binary feature vectors D, the *MTV* Error of encoding $\mathcal{E}$ is

$$-|D|H(\overline{\rho}_{\mathcal{E}}) + 1/2|\mathcal{E}| \log |D|$$

where $H(\overline{\rho}_{\mathcal{E}})$ is the entropy of maximum entropy distribution $\overline{\rho}_{\mathcal{E}}$ defined in Section 2.4.1. The second term in *MTV* Error penalizes Verbosity of the encoding $\mathcal{E}$. Since naive encoding assumes feature independence, we can first compute entropy

of the marginal distribution of each individual feature. Entropy $H(\overline{\rho}_{\mathcal{E}})$ is simply the sum of feature entropies.

Evaluating Naive Mixture Encoding. Evaluation of *naive encoding* can be generalized to *naive mixture* by taking a weighted average over resulting clusters (See Section 2.5.2).

2.8.1.2 Classical Laserlight and MTV

Establishing Baselines. To establish baselines, we evaluate *Laserlight* and *MTV* on their own data sets. The take-aways from related experiments are that (1) *naive encoding* is faster and more accurate than classical *Laserlight* and *MTV*; (2) the runtime increases superlinearly with the number of patterns mined from both *Laserlight* and *MTV*. For detailed experiment results, we refer the reader to [20].

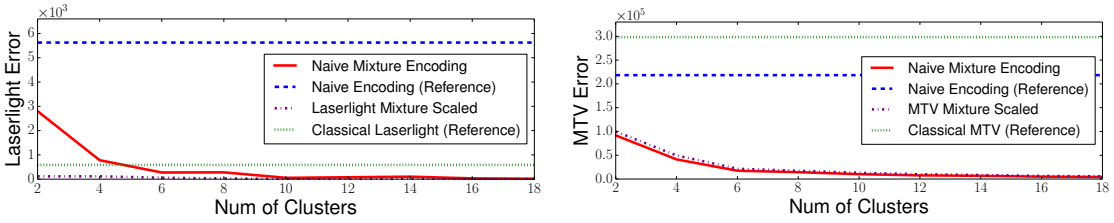
Anti-correlation and Dimentionality Reduction. Recall in Section 2.7.2.2 that *Laserlight* is restricted to 100 features. For its own *Income* data set, *Laserlight* can be applied with its full set of 783 features. This is due to the prior knowledge that the 783 features belong to 9 groups. In each group, features are mutually anti-correlated which can be reduced to a single feature. Similarly, *Mushroom* data set can be reduced from 95 to 21 features.

2.8.1.3 Generalizing Laserlight and MTV

We generalize *Laserlight* and *MTV* on partitioned data by applying them on each cluster. We then combine Errors on all clusters by taking a weighted average, as described in Section 2.5.2. Depending on how many patterns are mined from each cluster, *Laserlight* and *MTV* can be generalized into two types: (1) The number of patterns mined from each cluster is scaled to be equal to Verbosity of the *naive*

encoding; and (2) The total number of patterns mined from all clusters is fixed to a given number. We name the first type *Laserlight (MTV) Mixture Scaled*, which is comparable to *naive mixture encoding*. We name the second type *Laserlight (MTV) Mixture Fixed*, which is comparable to the classical *LaserLight (MTV)* algorithm.

Take-away. As the data is partitioned into more clusters, both runtime and Error of *Laserlight (MTV) Mixture Fixed* exponentially decrease. This observation can be potentially generalized to other pattern mining algorithms. For experiment details, we refer the reader to [20].



(a) **Laserlight Error v. # of Clusters on Mushroom data**

(b) **MTV Error v. # of Clusters on Mushroom data**

Figure 2.6: **Naive Mixture v. Laserlight/MTV Mixture**

2.8.1.4 Comparison with Naive Mixture Encoding

At last, we compare *Laserlight (MTV) Mixture Scaled* with *naive mixture encoding*. Note that it is time-consuming for *Laserlight* to mine the same number of patterns as *naive encoding* on *Income* data (See runtime analysis in [20]), we choose *Mushroom* data for *Laserlight Mixture Scaled* instead. The experiment results are given in Figure 2.6. The x-axis for all sub-figures in Figure 2.6 represents the number of clusters and the y-axes stands for *Laserlight* and *MTV* Error respectively. We incorporate baselines (i.e., *naive encoding*, classical *Laserlight* and *MTV*) as reference lines in Figure 2.6a and 2.6b respectively. We also experienced a limitation of 15 patterns for *MTV*. Hence the comparison between *MTV Mixture Scaled* and naive mixture encoding is not strictly on equal footing as *MTV Mixture Scaled* is not

able to reach the same Total Verbosity as *naive mixture encoding*. Note that their difference in Verbosity is mitigated by the fact that *MTV* Error measure penalizes encoding Verbosity.

Figure 2.6a shows that both *naive mixture encoding* and *Laserlight Mixture Scaled* have lower Error than their baselines. In addition, *Laserlight Mixture Scaled* has lower Error than *naive mixture encoding* when the number of clusters is less than 4 and they become close after 6 clusters. In other words, *Laserlight* is more accurate on lightly partitioned data. As the data is further partitioned, clusters become easier to summarize, and *naive encoding* becomes more similar to *Laserlight*. Figure 2.6b shows that *naive mixture encoding* marginally outperforms *MTV Mixture Scaled*.

Take-away. *Naive mixture encoding* is faster and has similar (lower) Error than *Laserlight (MTV) Mixture Scaled*.

2.9 Related Work

We aim at compressing query logs for accurately and efficiently computing workload statistics. Before the discussion of compression, we first review usecases and related work for workload analysis.

2.9.1 Workload Analysis

Existing approaches related to workload analysis usually aim at specific tasks like query recommendation [32, 33, 34, 35, 17], performance optimization [19, 36], outlier detection [37] or visual analysis [18].

Query Recommendation. This task aims at tracking historical querying behavior and generating query recommendations. Related approaches [32, 34] flatten a query

abstract syntax tree as a set of *fragments* [32] or *snippets* [34]. User profiles are then built by grouping and summarizing queries of specific users in order to make personalized recommendation. Under OLAP systems, profiles are also built for workloads of similar OLAP sessions [17].

Performance Optimization. Index selection [38, 39] and materialized view selection [40, 19, 36] are typical performance optimization tasks. The configuration search space is usually large, but can be reduced with appropriate summaries.

Outlier Detection. Kamra *et al.* [37] aim at detecting anomalous behavior of queries in the log by summarizing query logs into profiles of normal user behavior.

Visual Analysis. Makiyama *et al.* [18] provide a set of visualizations that facilitate further workload analysis on Sloan Digital Sky Survey (SDSS) dataset. QueryScope [41] aims at finding better tuning opportunities by helping human experts to identify patterns shared among queries.

In these approaches, queries are commonly encoded as feature vectors or bit-maps where a bit array is mapped to a list of features with 1 in a position if the corresponding feature appears in the query and 0 otherwise. Workloads under the bit-map encoding must then be compressed before they can be efficiently queried or visualized for analysis.

2.9.2 Workload Compression Schemes

Run-length Encoding. *Run-length encoding (RLE)* is a loss-less compression scheme commonly used in *Inverted Index Compression* [42, 43] and *Column-Oriented Compression* [44]. RLE-based compression algorithms include but not limited to: Byte-aligned Bitmap Code (BBC) used in Oracle systems [45], Word-aligned Hybrid (WAH) [46] and many others [47, 48, 49]. In general, RLE-based methods focus on

column-wise compression and requires additional heavyweight inference on frequencies of cross-column (i.e., row-wise) patterns used for workload analysis.

Lempel-Ziv Encoding. Lempel-Ziv [7, 8] is the loss-less compression algorithm used by gzip. It takes variable sized patterns (row-wise in our case) and replaces them with fixed length codes, in contrast to Huffman encoding [9]. Lempel-Ziv encoding does not require knowledge about pattern frequencies in advance and builds the pattern dictionary dynamically. There are many other similar schemes for compressing files represented as sequential bit-maps, e.g. [50].

Dictionary Encoding. *Dictionary encoding* is a more general form of Lempel-Ziv. It has the advantage that patterns with frequencies stored in the dictionary can be interpreted as workloads statistics useful for analysis. In this paper, we extend dictionary encoding and focus on using a dictionary to infer frequencies of patterns not in it. Mampaey *et al.* proposed *MTV* algorithm [15] that finds the dictionary (of given size) having optimal *Bayesian Information Criterion(BIC)* score. Gebaly *et al.* proposed *Laserlight* algorithm [16] that builds a pattern dictionary for correctly inferring the truth-value of some augmented binary feature.

Generative Models. A generative model is a lossy compressed representation of the original log. Typical generative models are *probabilistic topic models* [12, 13] and *noisy-channel* model [14]. Generative models can infer pattern frequencies but they lack a model-independent measure for efficiently evaluating overall inference accuracy.

Matrix Decomposition. Matrix decomposition methods including Principal Component Analysis (PCA) [11] and Non-negative matrix factorization (NMF) [10] offer lossy data compression. But the resulting matrices after decomposition are not suited for inferring workload statistics.

2.10 Conclusions

In this paper, we introduced the problem of log compression and defined a family of pattern-based log encodings. We precisely characterized the information content of logs and offered three principled and one practical measures of encoding quality: Verbosity, Ambiguity, Deviation and Reproduction Error. To reduce the search space of pattern-based encodings, we introduced the idea of log partitioning, which induces the family of pattern mixture as well as its simplified form: naive mixture encodings. Finally, we experimentally showed that naive mixture encodings are more informative and can be constructed more efficiently than state-of-the-art pattern-based summarization techniques. We expect that making accurate and efficient inference on pattern frequencies will enable a range of more powerful database tuning and intrusion detection systems.

2.11 Future Work

Multiplicity-aware clustering. As the number of feature vectors can be millions or more, practically we only keep *distinct* feature vectors as input of clustering schemes. We can store feature vector frequencies in a separate column called *multiplicities*. A multiplicity-ignorant clustering scheme assumes a uniform distribution of queries in the log. However, query distributions $p(Q)$ of production database logs are usually skewed. For example, routine queries repeat themselves overwhelmingly in the log but contribute to a minority of distinct queries. We plan to improve naive mixture encodings by exploring *multiplicity-aware* clustering schemes such that distinct feature vectors can be clustered *as if they have been replicated*. The use of mixture models for summarization has potential implications for work on pattern

mining; As we show, existing techniques can be substantially improved both in runtime and Error.

Feature Clustering. For the usecase of materialized view selection, computing pattern frequencies may not be enough. We may need to summarize a query log as a limited set of *basis* views such that queries in the log can be represented by a simple join of a subset of basis views. Capturing basis views is not only relevant to data tuning tasks, but also facilitates human inspection of workloads in the log. To achieve the goal, in addition to partitioning queries into separate workload clusters, for each cluster we need to further partition its features into separate clusters where each cluster is equivalent to a *basis view*.

2.12 Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful comments and contributions. This work was supported by NSF Awards SaTC-1409551 and IIS-1750460. The conclusions and opinions in this work are solely those of the authors and do not represent the views of the National Science Foundation.

Chapter 3

Similarity Metrics for SQL Query Clustering

3.1 Introduction

Database access logs are used in a wide variety of settings, including evaluating database performance tuning [51], benchmark development [52], database auditing [53], and compliance validation [54]. Also, many user-centric systems utilize query logs to help users by providing recommendations and personalizing the user experience [55, 56, 57, 58, 59, 60]. As the basic unit of interaction between a database and its users, the sequence of SQL queries that a user issues effectively models the user's behavior. Queries that are similar in structure imply that they might be issued to perform similar duties. Examining a history of the queries serviced by a database can help database administrators with tuning, or help security analysts to assess the possibility and/or extent of a security breach. However, logs from enterprise database systems are far too large to examine manually. As one example, a recent study of queries at a major US bank for a period of 19 hours found nearly 17 million SQL

queries and over 60 million stored procedure execution events [53]. Even excluding stored procedures, it is unrealistic to expect any human to manually inspect all 17 million queries per day.

Let us consider an analyst (call her Jane) faced with the task of analyzing such a query log. Jane might first attempt to identify some interesting query fragments and their aggregate properties. For example, she might count how many times each table is accessed or the frequency with which different classes of join predicates occur. Unfortunately, such fine-grained properties lack the context to clearly communicate how the data is being used, combined, and/or manipulated. To see the complete context, Jane must look at entire queries. Naively, she might look at all *distinct query strings* in the log. Even comparatively small production databases typically log hundreds or thousands of distinct query strings, making direct inspection impractical. Furthermore, it is unclear that distinct query strings are the right level of granularity in the first place. Consider the following example queries:

1. ```
   SELECT name FROM user
   WHERE rank IN ('adm','sup')
   ```
2. ```
   SELECT SUM(balance) FROM accounts
   ```
3. ```
   SELECT name FROM user WHERE rank = 'adm'
       UNION SELECT name FROM user
             WHERE rank = 'sup'
   ```
4. ```
   SELECT SUM(accounts.balance) FROM accounts
       NATURAL JOIN user WHERE user.rank = 'adm'
   ```

Queries 1 and 2 are clearly distinct: Their structures differ, they reference different datasets, and perform different computations. The remaining queries however are less so. Query 3 is logically equivalent to Query 1: Both compute identical results.

Conversely, although Query 4 is distinct from Queries 1 and 2, it is conceptually similar to both and shares many structural features with each.

The exact definition of similarity may depend on Jane's exact task, the content of the log, the database schema, database records, and numerous other details, some of which may not be available to Jane immediately when she first begins analyzing the log. It is also likely that some of this information, like the precise contents of the database or even the database schema may not even be available to Jane for reasons of privacy or security. As a result, this type of log analysis can quickly become a tedious, time-consuming process [61]. An earlier work of Aligon et al. [62] attempted to address this problem for OLAP operations by performing query log analysis and exploration. Within the scope of this article, we focus on analysis of SQL queries instead of OLAP queries. In particular, we lay the groundwork for a more automated approach to SQL query log exploration based on hierarchical clustering. Given a hierarchical clustering of the SQL query log, Jane can manually adjust how aggressively the log is summarized. She can select an appropriate level of granularity without a *priori* needing to specify exactly what constitutes a similar query.

The primary focus of this article is to study the suitability of three existing query distance metrics [63, 64, 65] to be used with hierarchical algorithms for clustering query logs. All of these metrics operate on the query structure and do not rely on the availability of underlying data or schema, thus making them applicable in a wide variety of practical settings. We evaluate the three metrics on two types of data: Human-authored and Machine-generated. Thus, using an appropriate similarity metric, one can cluster the queries to obtain a meaningful clustering of the query log.

For our evaluation, we use three evaluation data sets:

1. a large set of student authored queries released by IIT Bombay [1],
2. a smaller set of student queries gathered at the University at Buffalo, and released as part of this publication, and
3. SQL logs that capture all activities on 11 Android phones for a period of one month [52].

Student-written queries are appealing, as queries are already labeled by their ground-truth clusterings — For each question, the student is attempting to accomplish one specific stated task. Conversely, machine-generated queries on smartphones present a conceptually easier challenge, as they produce more rigid, structured queries. The three similarity metrics are evaluated on these data sets using three standard clustering evaluation statistics: Silhouette Coefficient, Beta CV, and Dunn Index [66].

None of the similarity metrics perform as well as desired, so we propose and evaluate a pre-processing step to create more regular, uniform query representations by leveraging query equivalence rules and data partitioning operations. These rules are commonly utilized by database management systems when parsing and evaluating SQL queries. This process significantly improves the quality of all three distance metrics. We also investigate and identify sources of errors in the clustering process. Experimental results show that our *regularization* pre-processing technique consistently improves clustering for different query comparison schemes from the literature.

Concretely, the specific contributions of this article are:

1. A survey of existing SQL query similarity metrics,
2. An evaluation of these metrics on multiple query logs, and
3. Applying query standardization techniques to improve query clustering accuracy.

This article is organized as follows. We start by performing a literature survey on log clustering and SQL query similarity in Section 3.2. We describe a feature engineering technique called regularization in Section 3.3. In Section 3.4, we explain our query workloads and propose a strategy for evaluating the quality of query similarity metrics. The evaluation is presented in Section 3.5. We discuss our experiment results, findings and ideas to further build upon the surveyed techniques in Section 3.6, and in Section 3.7, we explain how this work can be beneficial by giving real life examples. Finally, we conclude by identifying the steps needed to deploy query log clustering into practice using the techniques evaluated in this article in Section 3.8.

3.2 Background

Analyzing query logs mostly relies on the structure of queries [67], although their motivations are different; some methods prefer using the log as a resource to collect information to build user profiles, and the others utilize structural similarity to perform tasks like query recommendation [56, 57, 60], performance optimization [63], session identification [64] and workload analysis [65]. A summary of these methods is given in Table 3.1.

There are also other possible approaches; like data-centric query comparison [68], and utilizing the *access areas* of user queries by inspecting the data partition the query is interested in [69] from the `WHERE` condition. However, these approaches are out of our scope since we are interested in comparing and improving methods based on structural similarity; we assume that we do not have access to the data or the statistical information about the database.

Agrawal *et al.* [70] aim to rank the tuples returned by the SQL query based on

Paper title	Motivation	Features	Feature Structure	Distance Function	Similarity Ratio
Agrawal *et al.* (2006) [70]	Q. reply importance	Schema, rules	Vector	Cosine similarity	No
Giacometti *et al.* (2009) [56]	Q. recommendation	Difference pairs	Set	Difference query	No
Yang *et al.* (2009) [57]	Q. recommendation	Selection/join, projection	Graph	Jaccard coefficient on the graph edges	No
Stefanidis *et al.* (2009) [58]	Data Recommendation	Inner product of two queries	Vector	-	No
Khoussainova *et al.* (2010) [59]	Q. recommendation	Popularity of each query object	Graph	-	No
Chatzopoulou *et al.* (2011) [60]	Q. recommendation	Syntactic element frequency	Vector	Jaccard coefficient and cosine similarity	No
Aouiche *et al.* (2006) [63]	View selection	Selection/join, group-by	Vector	Hamming distance	Yes
Aligon *et al.* (2014) [64]	Session similarity	Selection/join, projection, group-by	3 Sets	Jaccard coefficient	Yes
Makiyama *et al.* (2016) [65]	Workload analysis	Term frequency of projection, selection/join, from, group-by and order-by	Vector	Cosine similarity	Yes

Table 3.1: SQL query similarity literature review

the context. They create a ruleset for contexts and evaluate the result of queries that belongs to the context according to the ruleset. They capture context and query as feature vectors and capture similarity through cosine distance between the vectors.

Chatzopoulou *et al.* [60] aim to assist non-expert users of scientific databases by tracking their querying behavior and generating personalized query recommendations. They deconstruct an SQL query into a bag of *fragments*. Each distinct fragment is a feature, with a weight assigned to it indicating its importance. Each

feature has two types of importance: (1) within the query and (2) for the overall workload. Similarity is defined upon common vector-based measures such as cosine similarity. A summarization/user profile for this approach is just a sum over all single query feature vectors that belong to their workload.

Yang *et al.* [57], on the other hand, build a graph following the query log by connecting associations of table attributes from the input and output of queries which are then used to compute the likelihood of an attribute appearing in a query with a similarity function like Jaccard coefficient. Their aim is again to assist users in writing SQL queries by analyzing query logs. Giacometti *et al.* [56], similarly, aim to make recommendations on the discoveries made in the previous sessions for users to spend less time on investigating similar information. They introduce *difference pairs* in order to measure the relevance of the previous discoveries. Difference pairs are essentially the result columns that is not included in the other return results; hence the method depends on having access to the data. Stefanidis *et al.* [58] takes a different approach, and instead of recommending candidate queries, they recommend tuples that may be of interest to the user. By doing so, the users may decide to change the selection criteria of their queries in order to include these results.

Sapia [55] creates a model that learns query templates to prefetch data in OLAP systems based on the user's past activity. SnipSuggest [59], on the other hand, is a context-aware SQL-autocomplete system that helps database users to write SQL queries by suggesting SQL snippets. In particular, it assigns a probability score to each subtree of a query based on the subtree's frequency in a query log. These probabilities are used to discover the most likely subtree that a user is attempting to construct, at interactive speeds.

Although these methods [70, 60, 57, 56, 58, 59] utilize query similarity one way or other to achieve their purpose, they don't directly offer a way to compare query

similarity. We aim to summarize the log and the most practical way to describe a query log is to group similar queries together so that we can provide summaries of these groups to the users. For this purpose, we need to be able to measure pairwise similarity between each query, hence we need a metric that can do so. As shown in Table 3.1, this condition is only satisfied by [63, 64, 65].

Aouiche *et al.* [63] is the first work we encountered that proposes a pairwise similarity metric between two SQL queries although it is not the aim of their work. They aim to optimize view selection in warehouses by the queries posed to the system. They consider the *selection*, *joins* and *group-by* items in the query to create vectors and use Hamming Distance to measure how similar two queries are. While creating the vector, it doesn't matter if an item appears more than once or where the item is. They cluster similar queries that creates a workload on the system and base their view creation strategy in the system on the clustering result.

Aligon *et al.* [64] study various approaches to defining a similarity function to compare OLAP sessions. They focus on comparing session similarity while also performing a survey on query similarity metrics. They identify *selection* and *join* items as the most relevant components in a query followed by the *group by* set. Inspired by the findings, they propose their own query similarity metric which considers *projection*, *group-by*, *selection-join* items for queries issued on OLAP datacubes. OLAP datacubes are multidimensional models, and they have hierarchy levels for the same attributes. Aligon *et al.* [64] measure the distance between the attributes on different hierarchy levels, and compute the set similarity for *projection*, *group-by*, and *selection-join* sets individually when comparing two queries. In our experiments, since we do not consider the hierarchy levels in an OLAP system but focus on databases, we consider all queries are on the same level in the schema to adjust the formulas presented in the paper. Namely, we compute set similarity of *pro-*

jection, *group-by*, *selection-join* sets of two queries with Jaccard coefficient. Also, Aligon *et al.* [64] provide the flexibility to adjust weights of the three feature sets based on the domain needs. We explore how the clustering quality is affected with various weightings in Appendix **??**.

Makiyama *et al.* [65] approach query log analysis with the goal of analyzing a system's workload, and they provide a set of experiments on Sloan Digital Sky Survey (SDSS) dataset. They extract the terms in *selection*, *joins*, *projection*, *from*, *group-by* and *order-by* items separately and record their appearance frequency. They create a feature vector using the frequency of these terms which they use to calculate the pairwise similarity of queries with cosine similarity. Instead of clustering, they perform the workload analysis with Self-Organizing Maps (SOM).

To further illustrate how the three structural metrics [63, 64, 65] work, we show the feature representations for the following query for each method in Table 3.2.

```
SELECT u.username, u.yearenrolled
FROM user u, accounts a
WHERE u.id = a.userid
  AND a.balance > 1000
  AND u.id > 20050001
GROUP BY u.yearenrolled
ORDER BY u.yearenrolled
```

In the next section, we propose a generalized feature engineering scheme for query comparison methods to improve the clustering quality. Our work evaluates the performance of the three methods [63, 64, 65] that directly describe a pairwise similarity metric in Section 3.4 due to the lack of performance evaluation for the query similarity metrics in the given studies. We also show that our feature engineering scheme improves the clustering quality with both statistical and empirical methods.

Paper title	Extracted Feature Vector
Aouiche *et al.* (2006) [63]	{'u.id', 'a.userid', 'a.balance', 'u.yearenrolled'}
Aligon *et al.* (2014) [64]	{'u.username', 'u.yearenrolled'} {'u.id', 'a.userid', 'a.balance'} {'u.yearenrolled'}
Makiyama *et al.* (2016) [65]	{'SELECT_u.username' →1, 'SELECT_u.yearenrolled' →1, 'FROM_user →1', 'FROM_accounts' →1, 'WHERE_u.id' →2, 'WHERE_a.userid' →1, 'WHERE_a.balance' →1, 'GROUPBY_u.yearenrolled' →1, 'ORDERBY_u.yearenrolled' →1}

Table 3.2: Representation of three similarity metrics

3.3 Feature Engineering

The grammar of SQL is declarative. By design, users can write queries in the way they feel most comfortable, letting well-established equivalence rules dictate a final evaluation strategy. As a result, many syntactically distinct queries may still be semantically equivalent. Recall example queries 1 and 3, paraphrased here:

1. `SELECT name FROM user`
   `WHERE rank = 'a' OR rank='s'`

3. `SELECT name FROM user WHERE rank = 'a'`
   `UNION SELECT name FROM user WHERE rank = 's'`

Though semantically distinct, these queries produce identical results for any input. Unfortunately similarity of results is not practical to implement: General query equivalence is NP-complete [71] for SQL92 and earlier, while SQL99 and later versions of SQL are turing-complete, due to the introduction of recursive queries.

However, we can still significantly improve clustering quality by standardizing

certain SQL features into a more regular form with techniques such as canonicalizing names and aliases, removing syntactic sugaring, and standardizing nested query predicates. This process of *regularization* aims to produce a new query that is more likely to be *structurally* similar to other *semantically* similar queries. Because the output is an ordinary SQL query, regularization may be used with any similarity metric. These process is similarly used in [72, 73], where Chandra *et al.* [72] generate mutations of SQL queries to catch diversions from a baseline query, and Sapia [73] creates OLAP query prototypes based on selected features and models user profiles.

Although the techniques we utilize for regularization are widely used in other settings, to the best of our knowledge, we introduce their usage to improve clustering quality. We also test all the techniques we use individually to find their impact on the regularization's overall effect. Our experiments in Section 3.5.2 show consistent improvements for all metrics evaluated in practical real world settings. In this section, we describe the transformations that we apply to regularize queries and the conditions under which they may be applied.

3.3.1 Regularization Rules

Canonicalize Names and Aliases. As we will show in our experiments in Section 3.5, table and attribute aliases are a significant source of error in matching. Consider the following two queries:

5. `SELECT name FROM user`

6. ```
   SELECT id
   FROM (SELECT name AS id FROM user) AS t
   ```

Although these queries are functionally identical, variable names are aliased in different ways. This is especially damaging for the three structural heuristics that
   ```

Before	After
b $\{>, \geq\}$ a	a $\{<, \leq\}$ b
x **BETWEEN** (a,b)	$a \leq x$ **AND** $x \leq b$
x **IN** (a,b,...)	x=a **OR** x=b **OR** ...
isnull(x,y)	**CASE WHEN** x is null **THEN** y **END**

Table 3.3: Syntactic Desugaring

we evaluate, each of which assumes that variable names follow a globally consistent pattern. Our first regularization step attempts to create a canonical naming scheme for both attributes and tables which is similar to one used in [72].

Syntax Desugaring. We remove SQL's redundant syntactic sugar following basic pattern-replacements as shown in Table 3.3.

EXISTS Standardization. Although SQL admits four classes of nested query predicates: (`EXISTS`, `IN`, `ANY`, and `ALL`), the `EXISTS` predicate is general enough to capture the semantics of the remaining operators [72]. Queries using the others are rewritten:

`x IN (SELECT y ...)` **becomes**

```
EXISTS (SELECT * ...WHERE x = y)
```

`x < ANY (SELECT y ...)` **becomes**

```
EXISTS (SELECT * ...WHERE x < y)
```

`x < ALL (SELECT y ...)` **becomes**

```
NOT EXISTS (SELECT * ...WHERE x ≥ y)
```

DNF Normalization. We normalize all boolean-valued expressions by converting them to disjunctive normal form (DNF). The choice of DNF is motivated by the ubiquity of conjunctive queries in most database applications, as well as by the natural correspondence between disjunctions and unions that we exploit below.

Commutative Operator Ordering. We standardize the order of expressions involving commutative and associative operators (e.g., $\wedge$, $\vee$, $+$, and $\times$) by defining a canonical order of all operands and traversing the expression tree bottom-up to ensure consistent order of all operands.

Flatten FROM-Nesting. We merge nested sub-queries in a `FROM` clause with its parent query as described in [72].

Nested Query De-correlation. A common database optimization called nested-query de-correlation [74] converts some `EXISTS` predicates into joins for more efficient evaluation. Note that this rewrite does not guarantee query result equivalence under *bag semantics* due to duplicated rows in the result. Hence we require that the parent query is either a `SELECT DISTINCT` or a duplicate-insensitive aggregate [75] (e.g. $max\{1,1\} = max\{1\}$, but $sum\{1,1\} \neq sum\{1\}$). If the `EXISTS` predicate is in a purely conjunctive `WHERE` clause, the de-correlation process simply moves the query nested in the `EXISTS` into the `FROM` clause of its parent query. The (formerly) nested query's `WHERE` clause can be then merged into the parent's `WHERE` clause. Specifically, if the input query is of the form:

```
SELECT ... FROM R WHERE
          EXISTS (SELECT ... FROM S WHERE q)
```

then the output query will have the form:

```
SELECT ... FROM R, (SELECT ... FROM S) WHERE q
```

To de-correlate a `NOT EXISTS` predicate, we use the set-difference operator `EXCEPT`. If the input is of the form:

```
SELECT DISTINCT... FROM R WHERE
      NOT EXISTS (SELECT ... FROM S WHERE q)
```

then the output will be of the form

```
(SELECT DISTINCT... FROM R) EXCEPT
  (SELECT DISTINCT... FROM R, WHERE
           EXISTS (SELECT ... FROM S WHERE q))
```

OR-UNION Transform. We use a regularization transformation that exploits the relationship between OR and UNION. This rewrite does not guarantee query result equivalence, also due to potentially duplicated rows in query result. Recall the equivalence between logical OR and UNION mentioned in our first example. Naively, we might convert the DNF-form predicates into UNION queries:

`SELECT ... WHERE q OR p OR ...` **becomes**

`SELECT ... WHERE q UNION SELECT ... WHERE p UNION ...`

However, duplicates caused by the possible correlation between clauses in DNF will break the equivalence of this rewrite. Consider the following query:

```
SELECT Score FROM Exam WHERE Score>60 OR Pass=1
```

Students who pass the exam overlap with those whose score greater than 60. Thus the rewritten query would not be exactly equivalent, as it may include duplicate rows. As a result, we require the query to satisfy the same condition mentioned in previous rule *nested query de-correlation*.

Union Pull-Out. Since the prior transformation may introduce UNION operator in nested subqueries, we push selection predicates down into the union as well.

3.4 Quality Metrics

In this section, we introduce the quality measures and workloads to evaluate three query similarity metrics and the feature engineering scheme. Our goal is to evaluate how well a query similarity metric captures the task behind a query with and without regularization. We use two types of real-world query workloads: human- and machine-generated. We expect the problem of query similarity to be harder

on human-generated workloads, as queries generated by machines are more likely to follow a strict, rigid structural pattern.

As a source of human-generated queries, we use two different sets of student answers to database course assignments. Many database courses include homework or exam questions where students are asked to translate prose into a precise SQL query. This provides us with a ground-truth source of queries with different structures that should be similar. As machine-generated queries, we use PocketData [52] a log of 33 million queries issued by smartphone apps running on 11 phones in the wild over the course of a month.

In subsection 3.4.1, we outline the datasets used. Then, in subsection 3.4.2, we outline the experimental methodology used to evaluate distance metrics, and propose a set of measures for quantitatively assessing how effective a query similarity metric is at clustering queries with similar tasks.

3.4.1 Workloads

We use three specific query sets: Student assignments gathered by IIT Bombay [1], student exams gathered at our department (denoted as UB dataset in the experiments) and released as part of this article[1], and SQL query logs of the Google+ app extracted from PocketData dataset [52].

The first dataset [1] consists of student answers to SQL questions given in IIT Bombay's undergraduate databases course. The dataset consists of student answers to 14 separate query-writing tasks, given as part of 3 separate homework assignments. The query writing tasks have varying degrees of difficulty. Answers are not linked to anonymous student identifiers and there is no grade information. The IIT Bombay dataset is exclusively answers to homework assignments, so we expect generally high-

[1] `http://odin.cse.buffalo.edu/public_data/2016-UB-Exam-Queries.zip`

quality answers due to the lack of time pressure and availability of resources for validating query correctness.

The second dataset consists of student answers to SQL questions given as part of our department's graduate database course. The dataset consists of student answers to 2 separate query-writing tasks, each given as part of midterm exams in 2014 and 2015 respectively. SQL queries were transcribed from hand-written exam answers, anonymized for IRB compliance and labeled with the grade the answer was given. We expect quality to vary, as exams are closed-book and students have limited time. Since 50% of the grade is the failing criterion, we assume that answers conform with the task of the question if the grade is over 50%. We also explore 20% and 80% thresholds in Appendix **??**.

The third dataset consists of SQL logs that capture all database activities of 11 Android phones for a period of one month. We selected Google+ application for our study since it is one of the few applications where all users created a workload. SQL queries collected were anonymized and some of the identified query constraints were deleted for IRB compliance [52].

A summary of all datasets is given in Tables 3.4, 3.5, and 3.6. The prose questions asked for IIT Bombay and UB Exam datatsets can be found in Table 3.7 and 3.8. Not all student responses are legitimate SQL, and so we ignore queries that cannot be successfully parsed by our open-source SQL parser[2]. We also released the source code we used in the experiments[3].

In the first two datasets, the query-writing task is specific. We can expect that student answers to a single question are written with the same task. Thus, we would expect a good distance metric to rate answers to the same question as close and

[2] `https://github.com/UBOdin/jsqlparser`
[3] `https://github.com/UBOdin/EttuBench`

Question	Total number of queries	Number of parsable queries	Number of distinct query strings
1	55	54	4
2	57	57	10
3	71	71	66
4	78	78	51
5	72	72	67
6	61	61	11
7	77	66	61
8	79	73	64
9	80	77	70
10	74	74	52
11	69	69	31
12	70	60	22
13	72	70	68
14	67	52	52

Table 3.4: Summary of IIT Bombay dataset

answers to different questions as distant. Similarly, using the distance metric for clustering, we would expect to see each query cluster to uniformly include answers to the same question.

In the third dataset, PocketData-Google+, the queries are generated by the Google+ application. Since some of the constants are replaced with standard placeholders for IRB compliance, the number of distinct queries drops significantly. Since there is no information about what kind of a task a query is trying to perform, we inspected and manually labeled each distinct query string. Queries were labeled with one of 8 different categories: Account, Activity, Analytics, Contacts, Feed, Housekeeping, Media and Photo.

Year	2014	2015
Total number of queries	117	60
Number of syntactically correct queries	110	51
Number of distinct query strings	110	51
Number of queries with score > 50%	62	40

Table 3.5: Summary of UB Exam dataset

	Pocket Dataset	Google+
All queries	45,090,798	2,340,625
SELECT queries	33,470,310	1,352,202
Distinct query strings	34,977	135

Table 3.6: Summary of PocketData dataset and Google+

3.4.2 Clustering validation measures

In addition to workload datasets, we define a set of measures to be used for evaluating queries. Given a set of queries labeled with tasks and an inter-query similarity metric, we want to understand how well the metric can (1) put queries that perform the same task close together even if they are written differently, and (2) differentiate queries that are labeled with different tasks.

We evaluate each metric according to how well it aligns with the ground-truth cluster labels. Rather than evaluating the clustering output itself, we evaluate an intermediate step: the pairwise distance matrix for the set of queries in a given workload. With this matrix and a labeled dataset, we can use various clustering validation measures to understand how effectively a similarity metric characterizes the partition of a set of queries. Specifically, clustering validation measures are used to validate the quality of a labeled dataset by estimating two quantities: (1) the degree of tightness of observations in the same label group and (2) the degree of separations between observations in different label groups. As a result, we will use

three clustering validation measures [66, Chapter 17] including Average Silhouette Coefficient, BetaCV and Dunn Index as they all quantify the two qualities mentioned above in their formulations.

Silhouette coefficient. For every data point in the dataset, its silhouette coefficent is a measure of how similar it is to its own cluster in comparison to other clusters. In particular, the silhouette coefficient for a data point i is measured as $\frac{b(i)-a(i)}{max(a(i),b(i))}$ where $a(i)$ is the average distance from i to all other data points in the same cluster and $b(i)$ is the average distance from i to all other data points in the closest neighboring cluster. The range of silhouette coefficient is from -1 to 1. We denote $s(i)$ to represent silhouette coefficient of data point i. $s(i)$ is close to 1 when $s(i)$ is close to other data points from the same cluster more than data points from different clusters, which represents a good match. On the other hand, $s(i)$ which is close to -1 represents that the data point i stayed in the wrong cluster, as it is closer to data points in different clusters than its own. Since the silhouette coefficient represents a measure of degree of goodness for each data point, to validate the effectiveness of the distance metric given a query partition, we use the average silhouette coefficient of all data points (all queries) in the dataset.

BetaCV measure. The BetaCV measure is the ratio of the total mean of intra-cluster distance to the total mean of inter-cluster distance. The smaller the value of BetaCV, the better the similarity metric characterizes the cluster partition of queries on average.

Dunn Index. The Dunn Index is defined as the ratio between minimum distance between query pairs from different clusters and the maximum distance between query pairs from the same cluster. In other words, this is the ratio between closest pairs of points from different clusters over the largest diameter among all clusters. Higher

values of the Dunn Index indicate better the worst-case performance of the clustering metric.

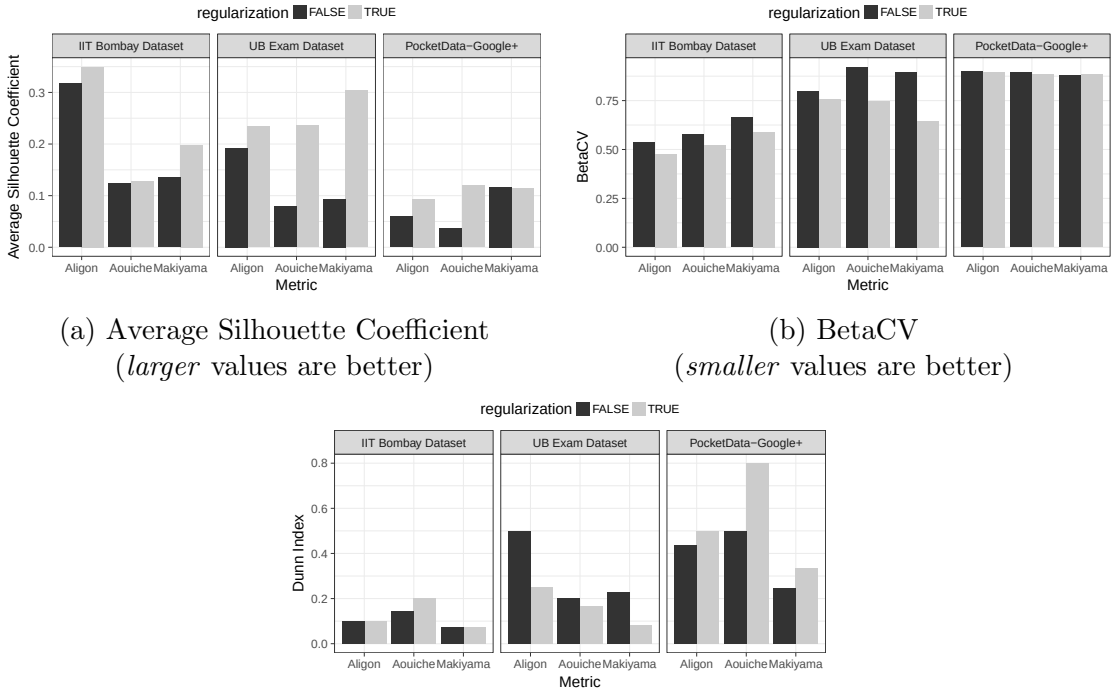
3.5 Experiments

In this section, we perform experiments to evaluate the performance of three similarity metrics previously discussed in Section 3.2: Makiyama's similarity [65], Aligon's similarity [64] and Aouiche's similarity [63]. We implemented each of these similarity metrics in Java and evaluated them using the three clustering validation measures discussed in subsection 3.4.2. In particular, we evaluate these three similarity metrics on their ability to capture the tasks performed by SQL queries. In addition, we also evaluate the effectiveness of the feature engineering step introduced in Section 3.3 and understand how query similarity can be improved by applying this step on the SQL query. We also look closer at feature engineering by breaking it down to different modules and analyze the effect of each module on capturing the tasks performed by queries.

3.5.1 Evaluation on SQL similarity metrics

In the first experiment, we evaluate three similarity metrics mentioned in Section 3.2. The aim of the experiment is to evaluate which similarity metric can best capture the task performed by each query.

The black columns in Figure 3.1 show a comparison of three similarity metrics using each of the three quality measures (Average Silhouette Coefficient, BetaCV and Dunn Index). As can be seen in Figure 3.1, Aligon seems to work the best for both IIT Bombay and UB Exam dataset while achieving second-best for PocketData-Google+ dataset under the Average Silhouette Coefficient measure. When consider-



(a) Average Silhouette Coefficient (*larger* values are better)

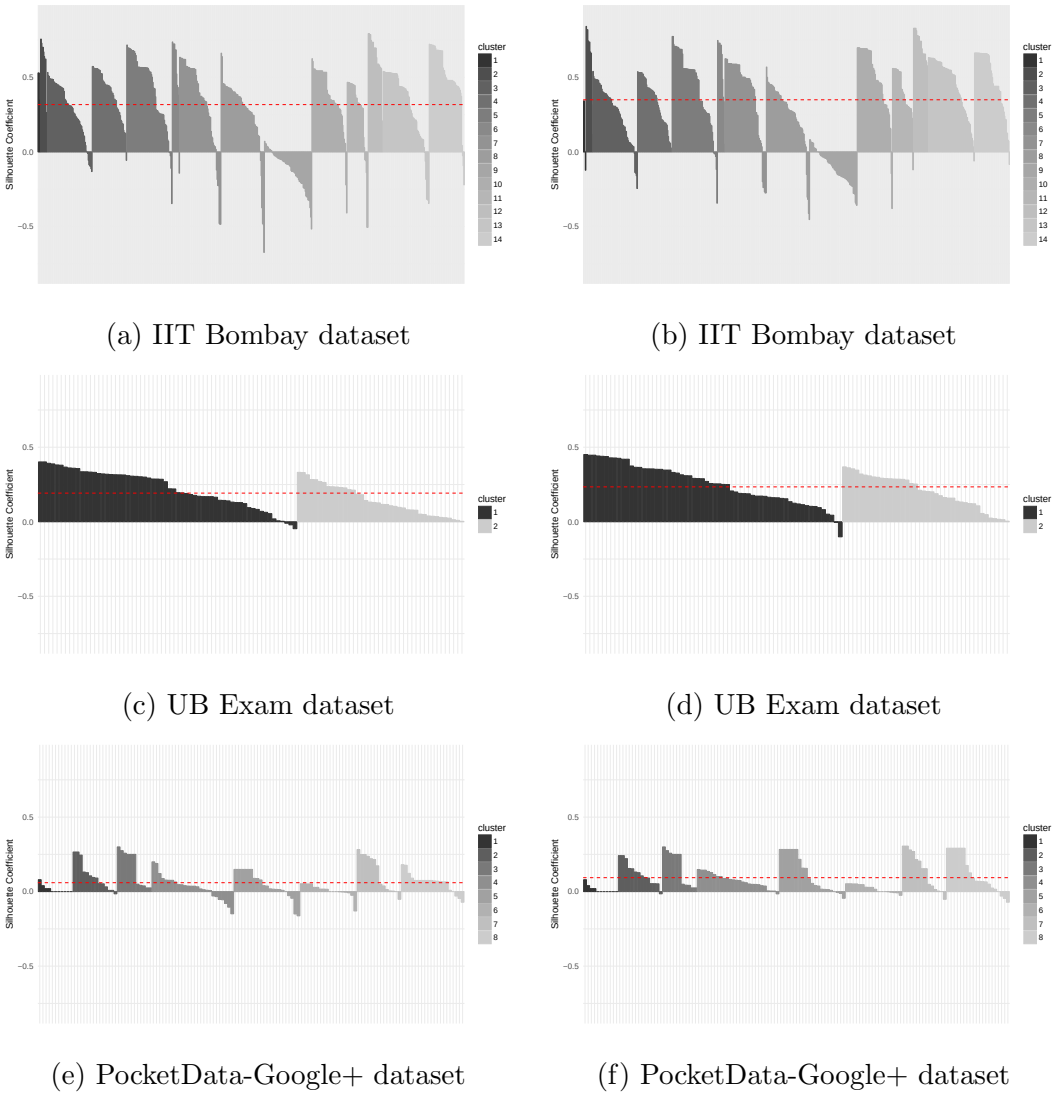
(b) BetaCV (*smaller* values are better)

(c) Dunn Index (*larger* values are better)

Figure 3.1: Clustering validation measures for each metric with and without regularization step

ing BetaCV measure, Aligon also attains the best result for both IIT Bombay and UB Exam dataset while having comparable result for PocketData-Google+ dataset. Aligon also performs well on the Dunn Index, coming in first on UB Exam dataset, and second-best for IIT Bombay and PocketData-Google+ dataset. Especially given that the Dunn Index measures only worst-case performance, Aligon's metric seems to be ideal for our workloads. This shows that even a fairly simple approach can capture task similarity well.

For a closer look of Aligon's similarity metric, Figure 3.2(a,c,e) shows the distribution of Silhouette coefficients for each query and their respective tasks. Recall that the silhouette coefficient below 0 effectively indicates a query closer to another cluster than its own, or a query that would be mis-classified. The further below zero,



(a) IIT Bombay dataset

(b) IIT Bombay dataset

(c) UB Exam dataset

(d) UB Exam dataset

(e) PocketData-Google+ dataset

(f) PocketData-Google+ dataset

Figure 3.2: Distribution of silhouette coefficients when using Aligon's similarity without regularization (a,c,e), and when regularization is applied (b,d,f)

the greater the error. For the UB Exam dataset (Figure 3.2c), the majority of queries would have been successfully classified, and only a small fraction exhibit minor errors. For the PocketData-Google+ dataset (Figure 3.2e), there are some erroneous queries in cluster 4, 5 and 6 while cluster 1, 2, 3, 7 and 8 have very few errors. For the Bombay dataset (Figure 3.2a), the distribution of errors varies. Cluster 1, 2, 4, 6, 12 and 14 exhibit virtually no error, while cluster 7, 8, and 9 exhibit particularly

egregious errors.

3.5.2 Evaluation of feature engineering

We next evaluate the effectiveness of regularization by applying it to each of the three metrics described in Section 3.2. We use our quality evaluation scheme to compare the quality of each measure both with and without feature engineering.

Figure 3.1 shows the values of three validation measures for each of the three similarity metrics, both with and without regularization. As shown in Figure 3.1, regularization significantly improves the Average Silhouette Coefficient and BetaCV measures for all similarity metrics except for the case of Makiyama similarity metric with PocketData-Google+ dataset. The Dunn index is relatively unchanged or little improved for the IIT Bombay and PocketData-Google+ dataset and shows slight signs of worsening with regularization on the UB Exam dataset. To understand the reason of worse Dunn Index, we compare Figure 3.2c (original) with Figure 3.2d (with regularization). The Silhouette Coefficient for answers that are originally positive in each question are considerably increased, and for answers that are originally negative (regarded erroneous) are even more decreased as a result of regularization, since it reduces the query structure diversity which leads to separating queries better. In other words, for erroneous answers with negative Silhouette Coefficients, distance metrics like Aligon distinguish them further apart from answers with positive Silhouette Coefficients after regularization. Since erroneous answers are treated as the 'worst cases' for each question, the Dunn Index which measures worst case performance naturally gets worse.

3.5.2.1 Per-Query Similarity

Figure 3.2(b,d,f) shows the distributions of silhouette coefficients for the Aligon similarity metric after regularization is applied. For IIT Bombay dataset, comparing against Figure 3.2a there is a slight improvement at the tail end of clusters 9, 11, 12, 13 and 14 — several of the negative coefficients have been removed. Furthermore, positive matches have been improved, particularly for cluster 7, 9, 10, 12 and 13. Finally, there has been a significant reduction in the degree of error in cluster 10. Cluster 10 is a particularly egregious case of aliasing, as the correct answer involves two self-joins in the same query. As a result, aliasing is a fundamental part of the correct query answer, and our rewrites could not reliably create a uniform set of alias names. In the UB Exam and PocketData-Google+ datasets, the improvement provided by regularization can be seen for queries with both positive and with negative values of $s(i)$.

3.5.3 Case Study

As part of our analysis, we attempted to provide empirical explanations for query errors, in particular for queries where $s(i) < 0$ for all three similarity metrics. Namely, we looked into the queries that are too far apart from the clusters they belong, and we categorized the reasons for misclassification based on these queries. We then investigated how the regularization process particularly affect these queries.

Almost all of these egregiously misclassified queries appear in the IIT Bombay dataset, the distribution of which is summarized in Table 3.9. The PocketData-Google+ dataset includes no egregiously misclassified queries, while the UB Exam dataset includes only one such query (which we tagged as a case of **Contextual equivalence**). We tagged each egregiously misclassified query with an explanation

that justifies why the query has a low s(i). Tags were drawn from the following list:

Ground-truth error. A student's response to the question may have been legitimately incorrect. This is a query that is correctly classified as an outlier. For example:

```
SELECT *
FROM (SELECT id, name, time_slot_id
 FROM (SELECT *
  FROM (SELECT *
   FROM student
    NATURAL JOIN takes) b1) a, section
   WHERE a.course_id = section.course_id) a1
```

This query was attempting to complete the task *"Find the ID and names of all students who have (in any year/semester) taken two courses in the same timeslot."*

Nested subquery. A student's response is equivalent to a legitimately correct answer but uses nested subqueries such that a heuristic distance metric cannot recognize. For example:

```
SELECT id, name FROM student
WHERE id IN (SELECT DISTINCT s.id
 FROM (SELECT * FROM takes NATURAL JOIN section) s,
  (SELECT * FROM takes NATURAL JOIN section) t
 WHERE s.id = t.id
  AND s.time_slot_id = t.time_slot_id
  AND s.course_id <> t.course_id)
```

Here, the subquery nesting structure is significantly different from other queries for of the same question.

Aliasing. Aliasing (e.g., `AS` in SQL) breaks a distance metric that relies on attribute and relation names. For example:

```
SELECT DISTINCT student.id, student.name
```

```
FROM student, takes, section AS a, section AS b
WHERE student.id = takes.id
  AND takes.course_id = a.course_id
  AND takes.course_id = b.course_id
  AND a.course_id <> b.course_id
  AND a.time_slot_id = b.time_slot_id
```

The student's use of `a` and `b` make this query hard to distinguish from other queries that may use other names for the attributes.

Insufficient features. Relevant query components are not sufficiently captured as features for a heuristic distance metric to distinguish between answers from sufficiently similar questions.

Too many features. Irrelevant query components create redundant features that artificially increase the distance between the query and cluster center. For example:

```
SELECT DISTINCT student.name, takes.id,
                s1.course_id, s2.course_id
FROM section AS s1, section AS s2, takes, student
WHERE takes.course_id = s1.course_id
  AND s1.course_id <> s2.course_id
  AND s1.time_slot_id = s2.time_slot_id
  AND s1.semester = s2.semester
  AND s1.year = s2.year
  AND takes.sec_id = s1.sec_id
  AND s1.semester = takes.semester
  AND s1.year = takes.year
  AND student.id = takes.id
  AND s2.time_slot_id = s2.time_slot_id
  AND takes.sec_id = s2.sec_id
  AND s2.semester = takes.semester
  AND s2.year = takes.year
```

Contextual equivalence. Establishing query equivalence to properly clustered queries requires domain-specific knowledge not available to the distance metric (e.g. attribute uniqueness). For example:

```
SELECT student.id, student.name
FROM student
WHERE student.id
  IN (SELECT takes.id
    FROM takes, section
    WHERE takes.course_id = section.course_id
      AND takes.sec_id = section.sec_id
      AND takes.semester = section.semester
      AND takes.year = section.year
    GROUP BY takes.id,
        takes.semester,
        takes.year,
        section.time_slot_id
    HAVING count(*) > 1)
```

Table 3.9 shows the primary reasons why these queries could not be classified correctly. Note that there may be more than one reason for a query to be placed in a different cluster, but in Table 3.9, we only give the empirically determined primary reason.

Many of the queries with low silhouette coefficients are identified as incorrect answers for the task given. These answers directly affect the ground-truth quality, therefore reduce the average silhouette coefficient. Another reason for erroneous queries with low silhouette coefficients is because of aliasing. Although it is convenient for user to use aliases in the query to refer to a particular item, it is difficult for a machine to approximate the tasks the query authors are trying to accomplish since different query authors have different ways to name particular items in the query.

This problem is particularly prevalent in question 9 of the IIT Bombay dataset.

Although the distribution of the error reasons are expected to change, all the tags provided in this section can generically be applied to other query logs given a ground-truth. The regularization method cannot be expected to fix errors originating from misclassifications in ground-truth since they do not actually share any similarities with the cluster.

After the regularization process, the silhouette coefficient under all three similarity metrics for each query is computed again and the result yields an 18% overall reduction in number of erroneous queries ($s(i) < 0$) in the IIT Bombay dataset.

3.5.4 Analysis of regularization by module

In Subsection 3.5.2, we analyzed the overall effect of regularization on query similarity. However, as described in Section 3.3, regularization is composed of many different transformation rules. In this experiment, we group these rules into four separate modules and inspect their impact on the clustering quality. One may observe that Commutative Operator Ordering is guaranteed to provide benefit in structure similarity comparison, hence we include it in all four modules. In addition, there are dependencies between rules that require them to operate one before another. For example, we should better apply Syntax Desugaring and then DNF Normalization to simplify the boolean expression in WHERE clause before OR-Union Transformation. As another example, Exists Standardization should better be applied on nested subqueries before we de-correlate them using Nested Query De-correlation. As a result, we group the rules from Section 3.3 into four modules:

1. *Naming*: **Canonicalize Names and Aliases**

2. *Expression Standardization*: **Syntax Desugaring**,

Exists Standardization, **DNF Normalization**,
Nested Query Decorrelation, **OR-Union Transform**

3. *FROM-Nesting*: **Flatten FROM-Nesting**

4. *Union Pullout*: **OR-UNION Pullout**

Commutative Operator Ordering is included in all modules.

Figure 3.3 provides a comparison of each module in regularization. From this figure, one can observe that, since students use different names/aliases for their convenience when constructing queries, the *Naming* module is the most effective one in terms of improving clustering quality for IIT Bombay and UB Exam datasets. On the other hand, for PocketData-Google+ dataset, names are already canonicalized as they are machine-generated. In this case, *Expression Standardization* seems to be the most effective module, especially when using Aligon or Aouiche as similarity metric. In PocketData-Google+ dataset, referred tables and boolean expressions in the queries are both informative in distinguishing between different query categories or clusters. For this reason, Makiyama similarity metric which considers both works well even without regularization while Aligon and Aouiche can get commensurate performance only after applying *Expression Standardization* module.

Note that in Figure 3.3, *Expression Standardization* makes Average Silhouette Coefficient worse in some cases for IIT Bombay and UB Exam data sets. The performance degradation is majorly due to *feature duplication*. More specifically, consider the example query with *Expression Standardization*.

Example 5. *Syntax Desugaring with OR-UNION transform*

1.
```
SELECT name FROM usr WHERE
rank IN {'admin','normal'}
```

2.
```
SELECT name FROM usr WHERE
rank = 'admin' OR rank = 'normal'
```

3.
```
SELECT name FROM usr
WHERE rank = 'admin'
UNION
SELECT name FROM usr
WHERE rank = 'normal'
```

Query (1) is transformed into (2) by syntax desugaring and then into (3) by OR-UNION Transform. From (1) to (2), feature `WHERE rank` has been replicated; From (2) to (3), features `SELECT name` and `FROM usr` have been duplicated. For expressions of the form: `X IN` $\{x_1, x_2, \ldots, x_n\}$, feature duplication becomes dominant when n grows large. In Figure 3.3, Aligon and Makiyama suffer from feature duplication brought by *Expression Standardization* in some cases while Aouiche does not. Because Aouiche records feature existence instead of occurrence in its vector. Although in some cases such as this, simply replacing feature occurence with existence solves the problem of feature duplication, feature occurence can also be a good indicator for the interests of the query. We believe this problem can be addressed with exploration of feature weighting strategies. Therefore, the problem of feature duplication will be further explored as a part of feature weighting strategies in our future work.

3.6 Discussion

We have reviewed several similarity metrics for clustering queries and focused on three syntax-based methods that offer an end-to-end similarity metric. The advantage of this preference is that, syntax-based methods do not require access to the data in the database or database properties. Considering that only logs are usually

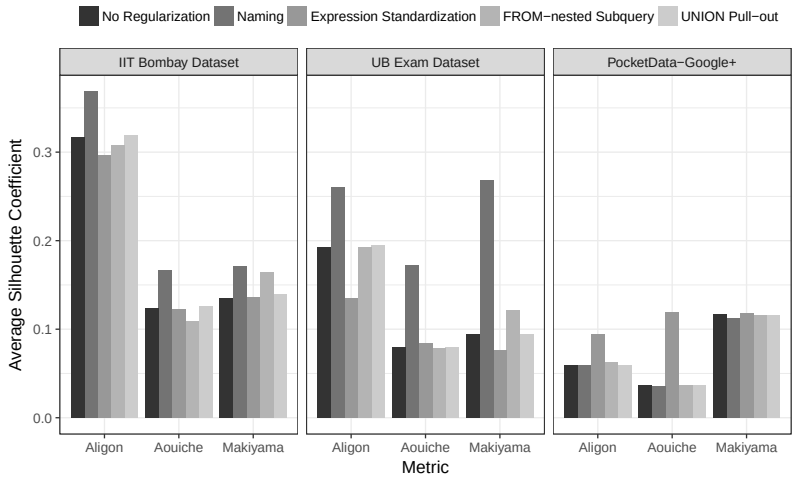


Figure 3.3: Effect of each module in regularization

transferred between organizations, and requiring access to the data for investigations can cause privacy violations, we preferred focusing on the syntax-based approach.

The survey we performed shows that most of the metrics make use of selection and join operations in the queries and consider them as the most important items for similarity calculation. Group-by aggregate follows them closely while projection items take the third most important item set. There are other possible feature sets that can be used, such as tables accessed or the abstract syntax tree (AST) of a query, but these feature sets are generally overlooked.

Although Aouiche *et al.* [63] make use of the most important features selection, joins, and group-by items, they don't utilize the number of times an item appears, or after the parsing, they don't consider what kind of feature an item is. This means, it does not matter if a query has `rank` column in group-by, and the other one has `rank` column in selection; they are considered the same. Makiyama *et al.* [65], on the other hand, follow Aligon *et al.* [64] in separating the different features, and improves on it by making use of appearance count of items. However, while trying to make use of every item like `FROM` and `Order-By` predicates, they consider these low priority

predicates with same importance as the selection and join predicates.

Makiyama *et al.* [65] use a more complete structure of the query AST, hence when the query is simple like in the PocketData-Google+ dataset, this technique can be slightly better. However, for a complex query with redundant features, mixing features captured from various components of a query without proper feature re-weighting will essentially decrease the weight of features that are more informative. Hence, in student exam datasets, we can observe that Aligon *et al.* [64] is better than the others while in PocketData-Google+ dataset, Makiyama *et al.* [65] is better.

We could further improve these methods by making use of the abstract syntax tree (AST) of a SQL statement. As a declarative language, the AST of a SQL statement acts as a proxy for the task of the query author. This suggests that a comparison of ASTs can be a meaningful metric for query similarity. For instance, we can group a query Q with other queries that have nearly (or completely) the same AST as Q. This structural definition of task has seen substantial use already, particularly in the translation of natural language queries into SQL [76]. For two SQL queries Q_1 and Q_2, one reasonable measure might be to count the number of connected subgraphs of Q_1 that are isomorphic to a subgraph of Q_2. Subgraph isomorphism is NP-complete, but a computationally tractable simplification of this metric can be found in the Weisfeiler-Lehman (WL) Algorithm [77, 53].

As can be seen in Tables 3.4 and 3.5, as the complexity or difficulty of the question increases, the number of distinct queries also increases, i.e., students find different ways to solve the same problem. Especially, in Table 3.5, no two students answer a question using the same structure. This phenomenon motivates the need for regularization in comparing SQL queries. As the complexity of the query increases, the possible ways to create the query to achieve the same task increase. Figure 3.1 shows that our assumption that regularizing queries will improve overall clustering

quality is correct. Our proposed feature engineering scheme improves the overall clustering quality of all three metrics on all three datasets, including both human- and machine-generated queries.

3.7 Application Scenarios

In this section, we provide three scenarios where the clustering scheme coupled with the proposed regularization is applicable:

The first one is, *Jane the DBA* where she takes on the task of improving database performance. After performing the straightforward database indexing tasks, she would need to select candidate *views*, which are virtual tables defined by a query. They allow querying just like tables by pre-fetching records from existing tables. Constructing a view for a frequent complex join operation can increase querying performance of the database substantially. To find the ideal views, Jane first clusters similar queries together to see what kinds of queries are more frequent. Making the most frequent complex query types faster by creating views of them could improve database performance substantially [63, 64].

The second one is, *Jane the security auditor* where she suspects that there is a person who leaks classified information from her organization. She can choose to investigate database access patterns along with other strategies which would involve query clustering [78]. After identifying the query clusters, she can partition the queries by the department or role to get the intuition about which departments and roles *normally* utilize what part of the database. She can detect the *outliers* from that behavior in order to determine the suspects for further investigation.

Lastly, *Jane the researcher* where needs to investigate the properties of the SQL query dataset that she is going to use for her research. One of the new graduate

students in her team clusters the queries, and provides her with the clustering assignments of each query. She doubts the quality of the clustering performed, and wonders if the clustering operation could be performed better.

Having a *better* clustering of queries would potentially enhance the quality of her work in all of the examples given above. Also, works cited in this section [63, 64, 78], along with many others can benefit from the framework described in this article.

3.8 Conclusion and Future Work

The focus of this work is to understand and improve similarity metrics for SQL queries relying on query structure to be used to cluster queries. We described a quality evaluation scheme that captures the notion of query task using student answers to query-construction problems and a real-world smartphone query load. We used this scheme to evaluate three existing query similarity metrics. We also proposed a feature engineering technique for standardizing query representations. Through further experiments, we showed that different workloads have different characteristics and no one similarity metric surveyed was always good. The feature engineering steps provided an improvement across the board because they addressed the error reasons we identified.

The approaches described in this article only represent the first steps towards tools for summarizing logs by tasks. Concretely, we plan to extend our work in several directions: First, we will explore new feature extracting mechanisms like the Weisfeiler-Lehman framework [53], feature weighting strategies and new labeling rules in order to capture the task behind logged queries better. Second, we will introduce the temporal order of the log to increase the query clustering quality. In this article, we focused on query structures to improve clustering quality. Exploring

the inter-query feature correlation based on query order can be used to summarize query logs in addition to clustering. Third, we will examine user interfaces that better present clusters of queries — Different feature sorting strategies in Frequent Pattern Trees (FP Trees) [79] in order to help the user distinguish important and irrelevant features, for example. Lastly, we will investigate the temporal effects on query clustering.

ID	Question
1	Find course_id and title of all the courses
2	Find course_id and title of all the courses offered by "Comp. Sci." department.
3	Find course_id, title and instructor ID for all the courses offered in Spring 2010
4	Find id and name of all the students who have taken the course "CS-101"
5	Find which all departments are offering courses in Spring 2010
6	Find the course ID and titles of all courses that have more than 3 credits
7	Find, for each course, the number of distinct students who have taken the course; in case the course has not been taken by any student, the value should be 0
8	Find id and title of all the courses offered in Spring 2010, which have no prerequisite
9	Find the ID and names of all students who have (in any year/semester) taken two courses
10	Find the departments (without duplicates) of courses that have the maximum credits
11	Show a list of all instructors (ID and name) along with the course_id of courses they have taught. If they have not taught any course show the ID and name with null value for course_id
12	Find IDs and names all students whose name contains the substring "sr" ignoring case. (Hint Oracle supports the functions lower and upper)
13	Using a combination of outer join and the is null predicate but WITHOUT USING "except/minus" and "not in" find IDs and names of all students who have not enrolled in any course in Spring 2010
14	A course is included in your CPI calculation if you passed it, or you have failed it, and have not subsequently passed it (or in other words, a failed course is removed from CPI calculation if you have subsequently passed it). Write an SQL query that shows all tuples of the relation other

Year	Question
2014	How many distinct species of bird have ever been seen by the observer who saw the most birds on December 15, 2013?
2015	You are hired by a local birdwatching organization, who's database uses the Birdwatcher Schema on page 2. You are asked to design a leader board for each species of Bird. The leader board ranks Observers by the number of Sightings for Birds of the given species. Write a query that computes the set of names of all Observers who are highest ranked on at least one leader board. Assume that there is no tied rankings.

Table 3.8: UB Exam dataset questions

Cause	**Erroneous queries Without Regularization**	**Erroneous queries With Regularization**
All queries	33 (100%)	27 (100%)
Ground-truth quality	14 (42.4%)	14 (51.8%)
Nested subquery	7 (21.2%)	5 (18.5%)
Aliasing	8 (24.2%)	5 (18.5%)
Insufficient features	2 (6.0%)	1 (3.7%)
Too many features	1 (3.0%)	1 (3.7%)
Contextual equivalence	1 (3.0%)	1 (3.7%)

Table 3.9: Empirical error reasons for IITBombay Dataset

Chapter 4

Application: Interactive Semi-Structured Schema Design

4.1 Introduction

Semi-structured formats like Json allow users to design schemas on-the-fly, as data is generated. For example, adding a new attribute to the output of a system logger does not break backwards compatibility with existing data. This flexibility facilitates the addition of new features and enables low-overhead adaptation of data-generating processes. However, because the data does not have a consistent underlying schema, it can be harder (and slower) to explore than simple tabular data. The logic of each and every query must explicitly account for variations in the schema like missing attributes. Performance also suffers, as there is no one physical data representation that is ideal for all schemas.

To address these problems, a variety of techniques [**?**, **?**, **?**, **?**, **?**] have arisen to generate schemas after-the-fact. The goal of these *semi-structured schema discovery* (S^3D) techniques is to propose a schema for collections of Json records. A common

approach to this problem is to bind the Json records to a normalized relational representation, or in other words, to derive a set of flat *views* over the hierarchical Json data.

Existing automated approaches to this problem (e.g., [**?**, **?**]) operate in a single-pass: They propose a schema and consider their job done. Unfortunately these techniques also rely heavily on general heuristics to select from among a set of schema design choices, as a clear declarative specification of a domain would be tantamount to having the schema already. To supplement domain-agnostic heuristics with feedback from domain experts, we propose a new *iterative and interactive* approach to S^{3}D called SchemaDrill.

SchemaDrill provides a OLAP-style interface (with analogs to roll-up, drill-down, and slice+dice) specialized for exploring collections of Json records. Every state of this interface corresponds to a relational view defined over the Json data. When ready, this view can be exported to a classical RDBMS or similar tool for simplified, more efficient data access. In this paper, we explore several design options for SchemaDrill, and discuss how each interacts with the challenges of S^{3}D.

4.1.1 Extracting Relational Entities

The first class of challenges we address involve the nuts and bolts of mapping hierarchical Json schemas to flat relational entities. Fundamentally, this involves a combination of three relational operators: Projection, Selection, and Unnesting.

Projecting Attributes. Json schema discovery can, naively, be thought of as a form of schema normalization [**?**], where each distinct path in a record is treated as its own attribute. Entities then, are simply groups of attributes projected out of the Json data, and the S^{3}D problem reduces to finding such groups (e.g., by using

Functional Dependencies [**?**]).

Selecting Records. This naive approach fails in situations where the collection of Json records is a mix of different entity types that share properties. As a simple example, Twitter streams mix three entity types: tweets, retweets, and deleted tweets. Although each entity appears in distinct records, they share attributes in common. Hence, entity extraction is not just normalization in the classical sense of partitioning attributes, but rather also a matter of partitioning records by content.

Collapsing Nested Collections. Json specifies two collection types: Arrays and Objects. Typically the former is used for encoding nested collections and the latter for encoding tuples with named attributes. However, this is not a strict requirement. For example, latitude and longitude are often encoded as a 2-element array. Conversely, in some data sets, objects are used as way to index collections by named keys rather than by positions. Hence, simple type analysis can not distinguish between the two cases. This is problematic because treating a collection as a tuple creates an explosion of attributes that make classical normalization techniques incredibly expensive.

4.1.2 Human-Scale S^{3}D

Even in settings where Json data is comparatively well behaved, it is common for it to have dozens, or even hundreds of attributes per record. Similarly, individual Json records can be built from any of the hundreds or thousands (or more) different permutations of the full set of attributes used across the entire collection. Bringing this information down to human scale requires simultaneously simplifying and summarizing.

Summarization. For the purposes of entity construction, the full set of attributes

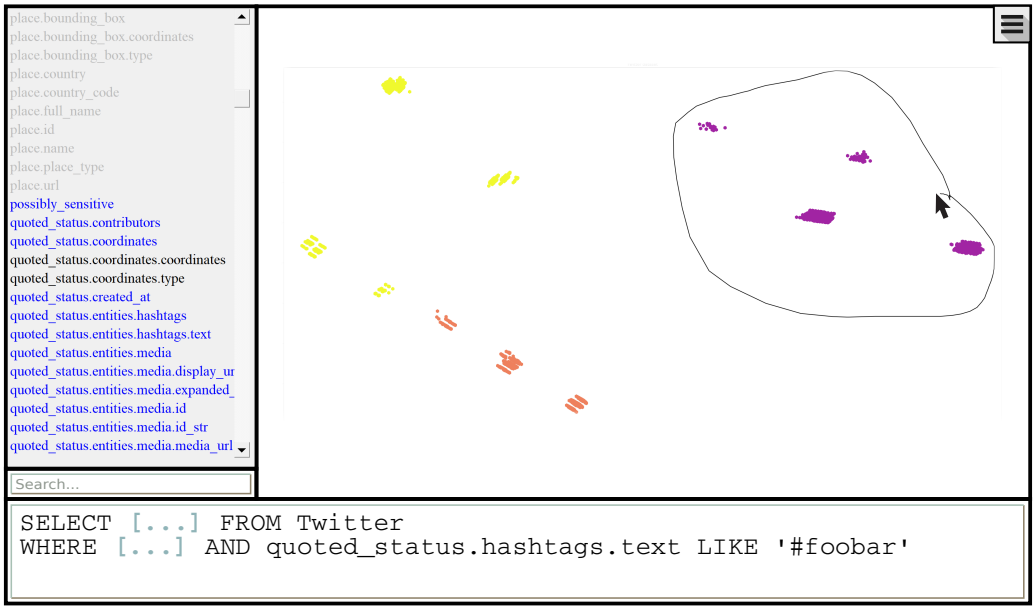


Figure 4.1: Prototype user interface for SchemaDrill.

is often unnecessary. It is often possible to collapse multiple attributes together, or express attributes as equivalent alternatives. As an example, an address might consist of four distinct attributes city, zip code, street, and number when it could conceptually be expressed as just one.

Visualization. In addition to simplifying the underlying problem, it is also useful to give users a coarse "top-down" view of the schema process. Specifically, users need to (1) be able to see patterns of structural similarity between distinct schemas, and (2) understand how much variation exists in the data set as-is.

Iteration. By combining straightforward summarization and data visualization techniques, SchemaDrill helps users to quickly identify natural clusters of records and attributes that represent relational entities. SchemaDrill facilitates an *iterative* schema design process to allow human experts to better evaluate whether structures in the data indicate conceptual relationships between records or attributes, or are merely data artifacts.

4.1.3 Overview

Figure 4.1 shows the prototype interface of SchemaDrill. The pane on the left, discussed in Section 4.2, shows the schema of the currently selected JSON view, highlighting attributes and groups based on relevance. The pane on the right, discussed in Section 4.3, provides a top-down visual sketch of schemas in the currently selected JSON data, and allows users to interactively filter out parts of it. Finally in Section 4.3, we show examples of how SchemaDrill facilitates incremental, iterative exploration and mapping of JSON schemas.

4.2 Summarization

The first component of SchemaDrill, the schema pane, shows the relational schema of the extracted view. Initially, this schema consists of one attribute for every path in the Json collection being summarized. Attributes may be deleted or restored, and sets of attributes may be unified.

The core challenge behind implementing this pane is that, depending on data set, the schema could consist of hundreds or thousands of attributes. This can be overwhelming for users who just wants to find account profiles appearing in a Twitter stream. To mitigate this problem, SchemaDrill presents the schema in a summarized form.

Specifically, attributes are grouped based on both correlations and anti-correlations between them. Groups of *correlated* attributes, or those that frequently co-occur in Json records are likely to be part of a common structure. Similarly, groups of *anti-correlated* attributes, or those that rarely co-occur in Json records are likely to represent alternatives (e.g., a Street Address vs GPS coordinates). We use correlations and anti-correlations between attributes to compact the schema for representation.

Before describing the summary itself, we first formalize the problem.

4.2.1 Data Model

A Json object is an order-independent mapping from keys to values. A key is a unique identifier of a Json object, typically a string. A value may be atomic or complex. Atomic values in Json may be integers, reals, strings, or booleans. A complex value is either a nested object, or an *array*, an indexed list of values. For simplicity, we model arrays as objects by treating array indexes as keys. A Json record may be any value type, but for simplicity of exposition we will assume that all records are objects.

Example 6. *A fragment of Twitter Tweet encoded as Json*

```
"tweet": {
  "text": "#SIGMOD2018 in Houston this year",
  "user": {
    "name" : "Alice Smith",  "id" : 12345,
    "friends_count": 1023, ...
  },
  "retweeted_status": {
    "tweet": {
      "user": {...}, "entities": {...},
      "place": {...}, "extended_entities": {...}
    },...
  },
  "place": {...}, "extended_entities": {...},
  "entities": { "hashtags": [ "SIGMOD2018" ],... },
 ...
```

```
}
```

Json objects are typically viewed as trees with atomic values at the leaves, complex values as inner nodes, and edges labeled with the keys of each child. Our goal is to identify commonalities in the structure of this tree across multiple Json records in a collection. To capture the structure, we define the *schema* S of a Json record as the record with all leaf values replaced by a constant $\perp$. A *path* P is a sequence of keys $P = (k_0, \ldots, k_N)$. For convenience, we will write paths using dots to separate keys (e.g., `tweet.text`). We say that a path appears in a schema (denoted $P \in S$) if it is possible to start at the root of S and traverse edges in order. If the value reached by this traversal is $\perp$, we say that P is a terminal path of S (denoted $P \perp S$).

4.2.2 Paths as Attributes

Ultimately, our goal is to create a flat, relational representation suitable for use with an entire collection of Json records. The first step to reaching this goal is to flatten individual Json schemas into collections of attributes. We begin with a naive translation where each attribute corresponds to one terminal path in the schema. We write $S^{\perp}$ to denote the path set, or relational schema of Json schema S, defined as: $S^{\perp} = \{\ P \mid P \perp S\ \}$ Since keys are unique, commutative and associative, this representation is interchangeable[1] with the tree representation. Hence, when clear from context we will abuse syntax, using S to denote both a schema and its path set.

Example 7. *The path set of the Json object from Example 6 includes the paths:* *1.* `tweet.text` *2.* `tweet.user.friends_count` *3.* `tweet.user.id` *4.* `tweet.entities.hashtags.[0]`

[1] modulo empty objects or arrays

Each terminal appears in the set. Note in particular that single element of the array at `tweet.entities.hashtags` *is assigned the key* `[0]`.

Path sets make it possible to consider containment relationships between schemas. We say that S_1 is contained in S_2 iff $S_1^\perp \subseteq S_2^\perp$.

4.2.3 Schema Collections

We now return to our main goal, summarizing the schemas of collections of Json records. The starting point for this process is the schemas themselves. Given a collection of Json records, we can extract the set of schemas $\{ S_1, \ldots, S_N \}$ of records in the collection, which we call the *source schemas*. One existing technique for summarizing these records, used by Oracle's JSON Data Guides [**?**, **?**], is to simply present the set of all paths that appear anywhere in this collection. We call this the *joint schema* $\mathbb{S}$:

$$\mathbb{S}^\perp \stackrel{def}{=} \bigcup_i S_i^\perp$$

Observe that, by definition, each of the source schemas is contained in the joint schema. The joint schema mirrors existing schemes for relational access to JSON data like for example. However, the joint schema can still be very large, with hundreds, thousands, or even tens of thousands of columns[2]. To summarize them we need an even more compact encoding for sets of schemas.

A Schema Algebra. As a basis for compacting schema sets, we define a simple algebra. Recall that we are particularly interested in summarizing cooccurrence and anti-cooccurence relationships between attributes.

$$\mathbf{A} \;:=\; \mathbf{P} \;\mid\; \emptyset \;\mid\; \mathbf{A} \wedge \mathbf{A} \;\mid\; \mathbf{A} \vee \mathbf{A}$$

[2]One dataset [**?**] achieved 2.4 thousand paths through nested collections of objects.

Expressions in the algebra construct sets of schemas from individual attributes. There are two types of leaf expressions in the algebra: A single terminal path P represents a singleton schema ($\{\{P\}\}$), while $\emptyset$ denotes a set containing no schemas $\{\}$. Disjunction acts as union, merging its two input sets:

$$\{ S_1, \ldots, S_N \} \vee \{ S'_1, \ldots, S'_M \} \stackrel{def}{=} \{ S_1, \ldots, S_N, S'_1, \ldots, S'_M \}$$

Disjunction models anti-correlation: The resulting schema set is effectively a collection of schema alternatives. For example, $P_1 \vee P_2$ indicates two alternative schemas: $\{P_1\}$ or $\{P_2\}$. Conjunction combines schema sets by cartesian product:

$$\{ S_1, \ldots, S_N \} \wedge \{ S'_1, \ldots, S'_M \} \stackrel{def}{=} \{ S_i \cup S'_j \mid i \in [1, N], j \in [1, M] \}$$

Conjunction models correlations: The resulting schema set mandates that exactly one option from the left-hand-side and one option from the right-hand-side be present. On singleton inputs, the result is also a singleton. For example, $P_1 \wedge P_2$ is a single schema that includes both P_1 and P_2. For inputs larger than one element, the conjunction requires one choice from each input. For example, $(P_1 \vee P_2) \wedge (P_3 \vee P_4)$ is the set of all schemas consisting of one of P_1 or P_2, and also one of P_3 or P_4.

Although we omit the proofs for conciseness, both $\wedge$ and $\vee$ are commutative and associative, and $\vee$ distributes over $\wedge$[3]. For conciseness, we use the following syntactic conventions: (1) When clear from context, a schema S denotes its own singleton set, and (2) We write $P_1 P_2$ to denote $P_1 \wedge P_2$.

This schema algebra gives us a range of ways to represent schema sets. At one extreme, the set of source schemas arrives in what is effectively disjunctive normal form (DNF). One schema may be expressed as a conjunction of its elements, and the

[3]To be precise, the structure $\langle \{\{\mathbf{P}\}\}, \vee, \wedge, \emptyset, \{\{\}\} \rangle$ is a semiring.

full set of source schemas can be constructed by disjunction. For example, the source schemas $\{P_1, P_2\}$ and $\{P_2, P_3\}$ may be represented in the algebra as $P_1P_2 \vee P_2P_3$.

At the other extreme, the joint schema is a superset of all of the source schemas. It too can be thought of as a schema set, albeit one that loses information about which attributes appear in which schemas. Hence, this joint schema set may be defined as the power set of all attributes in the joint schema.

$$2^{\mathbb{S}} = \bigwedge_{P \in \mathbb{S}} (P \vee \emptyset)$$

Observe that at a minimum, each of the source schemas must appear in this schema set ($S_i \in 2^{\mathbb{S}}$). However many other schemas appear in the resulting schema set as well.

4.2.4 Summarizing Schema Collections

These two extreme representations (the raw source schemas and the joint schema set) are bad, but for subtly different reasons. In both cases, the representation is too verbose. In the former case verboseness stems from redundancy, with significant overlap in variables between the source schemas. Conversely in the latter case it stems from imprecision, as the schema set encompasses schemas that do not appear in the source schemas. Of the two, the latter is more compact, in particular because each attribute appears no more than once. This is a distinct representational advantage because the joint schema can be displayed simply as a list.

We would like to preserve this only-once property. Our aim then, is to derive an algebraic expression (1) in which each attribute appears exactly once, and (2) that is as tight a fit to the original source schema set as possible. Ultimately, this problem reduces to polynomial factorization and the discovery of read-once formulas [**?**], a

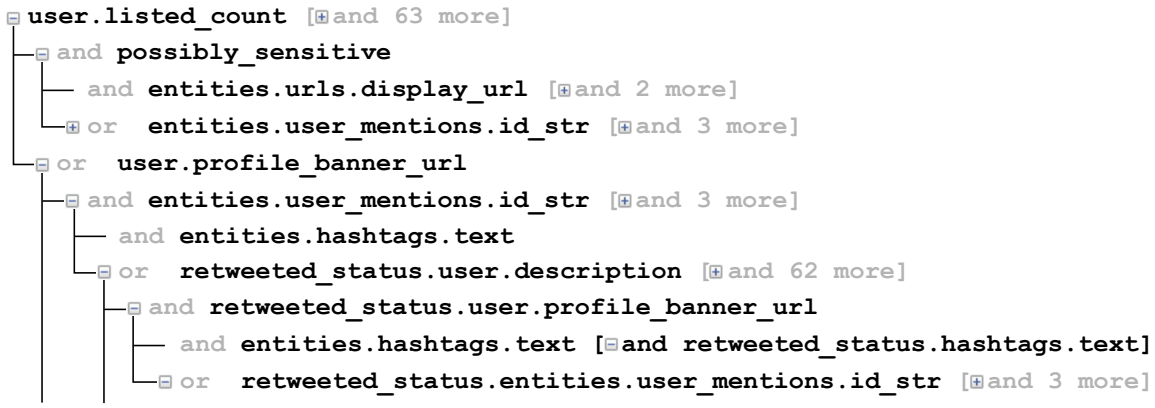


Figure 4.2: FP-Tree based schema summaries

problem that is, in general, worse than P-time [**?**]. Hence, for this paper, approximations are required. We consider two approaches and allow the user to select the most appropriate one for their needs. The first is based on Frequent Pattern Trees [**?**] (FPTrees), a data structure commonly used for frequent pattern mining. The second is to limit our search to read-once conjunctions of disjunctions of conjunctions, a form we call RCDC.

FP-Tree Summaries. An FP-Tree is a trie-like data structure that makes it easier to identify common patterns in a query. Every edge in the tree represents the inclusion of one feature, or in our case one attribute. Hence, every node in the tree corresponds to a set of paths (obtained by traversal from the root), and every leaf corresponds to one source schema. We observe that every node in an FP tree corresponds to a disjunction: For a node with 3 children, each subtree represents a different branch. Similarly, every edge corresponds to a conjunction with a singleton. Although the resulting tree may duplicate some attributes, duplications are minimized [**?**].

Example 8. *Figure 4.2 illustrates a schema summary based on FP-Trees. Sequences of nodes with a single child are collapsed into single rows of the display*

(e.g., `user.listed_count` *and 63 immediate descendents). A toggle switch allows these entities to be displayed to the user, if desired. Every level of the tree represents a set of alternatives. For example,* `possibly_sensitive` *never co-occurs with* `user.profile_banner_url`*.*

RCDC Summaries. Our second visualization is based on correlations and anticorrelations. To construct this visualization, we begin with the joint summary. Recall that the joint summary has the form

$$(P_1 \vee \emptyset)\ (P_2 \vee \emptyset)\ (P_3 \vee \emptyset)\ (P_4 \vee \emptyset) \ldots$$

We create a covariance matrix based on the probability of two attributes co-occurring in the schemas of one of our input Json records. Using this covariance matrix, hierarchical clustering [**?**], and a user-controlled threshold on the covariance, we cluster the attributes by parenthesizing. For example, clustering might group P_1 with P_2 and likewise P_3 with P_4. We can rewrite this formula as:

$$\approx (P_1P_2 \vee \emptyset)\ (P_3P_4 \vee \emptyset) \ldots$$

Observe that this formula omits schemas that the original formula captures (e.g., any schema including P_1 but not P_2). However, because clustering ensures that attributes within a group co-occur frequently, there are comparatively few such schemas.

We next repeat the process with a new covariance matrix built using the frequency of co-occurrence of *groups* (like P_1P_2). As before, we create clusters, but this time we cluster based on extremely negative co-variances. Hence, members of the resulting clusters are unlikely to co-occur. Continuing the example, let us assume that P_1P_2 and P_3P_4 are highly anti-correlated. Approximating and simplifying, we get an

```
user.listed_count [⊞and 63 more]
entities.user_mentions.id_str [⊞and 3 more] [⊞or 3 more]
possibly_sensitive [⊟]
 or user.profile_banner_url
retweeted_status.user.description [⊞and 66 more]
```

Figure 4.3: RCDC based schema summaries

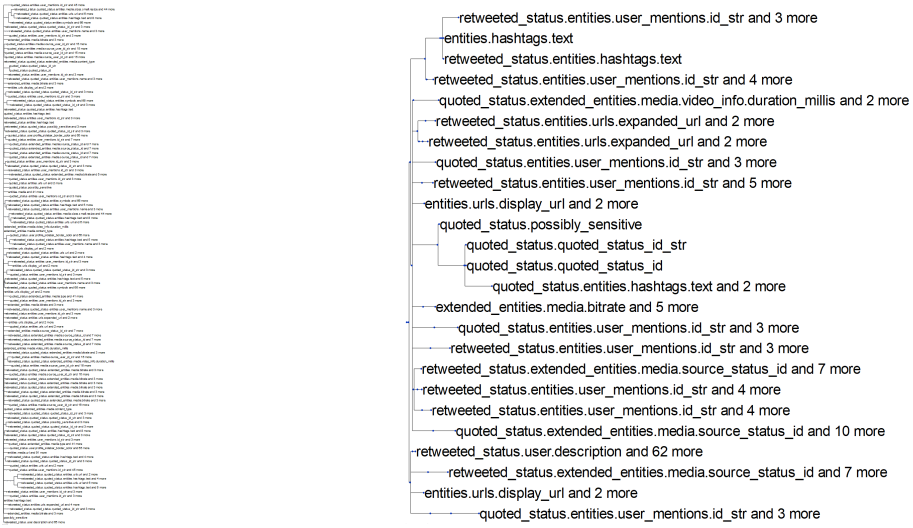
expression in RCDC form.

$$\approx (P_1P_2 \vee P_3P_4 \vee \emptyset) \ldots$$

As with the FP-Tree display, we use counts and an example attribute as a summary name for the group, and a toggle button to allow users to expand the group along either the OR or AND axes.

4.3 Visualization

Even within a mostly standardized collection of records like exported Twitter or Yelp data or production system logs, it is possible to find a range of schema usage patterns. Grouping by [anti-]cooccurrence is one step towards helping users understand these usage patterns, but is insufficient for three reasons: 1. Conceptually distinct fragments of the schema may share attributes in common (e.g., delete tweet records share attributes in common with tweet records). 2. Even if they do not co-occur, certain [groups of] attributes may be correlated (e.g., due to mobile phones, tweets with photos are also often geotagged). 3. There is no general way to differentiate Json objects and arrays being used to represent collections from those being used to represent structures (e.g., twitter stores geographical coordinates as a 2-element array). The second part of the SchemaDrill interface addresses these issues by pre-



(a) Without segmentation. (b) With segmentation.

Figure 4.4: Segmentation breaks up schema representations into manageable chunks.

senting top-down visual surveys of the schema. These surveys help users to quickly assess variations in schema usage across the collection, to identify related schema structures, and to "drill down" into finer grained relationships.

4.3.1 Schema Segmentation

Specifically, we want to help the user to focus on particular parts of the joint schema; We want to allow the user to filter out, or segment the schema based on certain required attributes that we call *subschemas*. We define a subschema s as a set of attributes, where s is contained in one or more source schemas. Further, the s-segment of source schemas $S_1, \ldots, S_N$ to be the subset that contain s:

$$\textbf{segment}(s) \stackrel{def}{=} \{\ S_i \mid i \in [1, N] \wedge s \subseteq S_i\ \}$$

We are specifically interested in visual representations that can help users to identify subschemas of interest. By then focusing solely on the segments defined by these

subschemas can significantly reduce the complexity of the schema design problem, as illustrated in Figure 4.4. Figures 4.4a illustrates the full schema summary as a tree, while Figure 4.4b shows a partial summary identified by the user using the lasso tool we describe shortly.

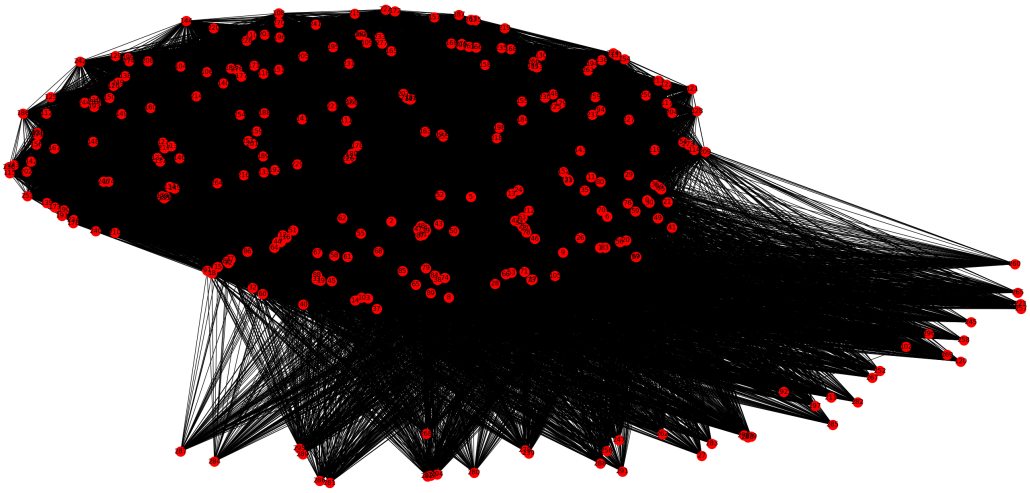


Figure 4.5: Coveriance Cloud for the full Yelp dataset.

Covariance Clouds. Our second visual representation, also like schema summaries, uses correlations and anti-correlations to communicate subschemas of interest. To generate a covariance cloud, we create a covariance matrix from the source schemas, using the appearance of each attribute as a variable. Based on a user-controllable threshold, we then construct a graph from the covariance matrix with every attribute as one node, and every covariance exceeding the threshold as an edge. The graph is then displayed to the user as a cloud using standard force-based layout techniques (e.g., those used by GraphViz [**?**]). Cliques in the graph represent commonly co-occurring subschemas that might form segments of interest. This includes every conjunctive group identified in the schema summary. However, unlike the schema summary, this visual representation more effectively captures subschemas with at-

tributes in common.

KNN-PCA Clouds. While the first visualizaton works on simple schemas, we found that on more complex Json data like Twitter streams [**?**], or the Yelp open dataset [**?**] there were too many inter-attribute relationships, and the resulting visualizatons were noisy. An approach we settled on is a mixture of Principle Component Analysis (PCA) and K Nearest Neighbor clustering (KNN). As before, we treat each source schema as a feature vector with each attribute representing one feature. We then use PCA to plot our source schemas in two-dimensions. The resulting visualization illustrates relationships between source schemas, with greater distances in the visualization representing (approximately) more differences between the schemas. Hence, clustered groupings of schemas represent potentially interesting sub-schemas.

A key limitation with this visualization is that for more complex datasets the somewhat arbitrary choice of 2 dimensions can be too low. Conversely adding more dimensions directly through PCA makes the visualization more complicated and hard to follow. To mitigate these limitations, we use K-Nearest Neighbors (KNN) to colorize the PCA Cloud. In addition to using PCA, we do KNN clustering on the schemas using a user-provided K (number of clusters). Each cluster identified by PCA is assigned a different color. Combined, these two algorithms to provide users an initial insight into the potential correlations that exist in their dataset.

Example 9. *Figures 4.1 and 4.6 show an example of the resulting view on Twitter and Yelp data. Note the much tighter clustering of attributes in the Twitter data: each cluster represents one particular, common type of tweet. Conversely, the Yelp schema includes a nested collection with, for example, hourly checkins at the business. The presence (or absence) of these terminal paths is more variable, and the resulting PCA cloud follows more of a gradient.*

Figure 4.6: KNN-PCA cloud with K=6 on the Yelp dataset.

These graphs may not initially be intuitive to interpret, but they provide abstract bearings that map directly to phenomenon's that exist in the data. In lieu of a formal user study, we postulate that the information gained from each algorithm independently will benefit the exploration process that is agnostic to labeled information. This is in contrast to other tools [**?**], that largely utilize naming cues and attribute nesting to meet user needs.

4.3.2 Schema Exploration

Now that we have shown the user potentially interesting subschemas, our next task is to help them to (1) narrow down on actually interesting subschemas, and (2) use those schemas to drill down to a segment of the schema data. For the first step, it is critical that the user develop a good intuition for what the visual representations encode. One way to accomplish this is to establish a bi-directional mapping between

SchemaDrill's two data panes.

To map from visual survey to schema summary, we provide users with a lasso tool. As illustrated in Figure 4.1, users can select regions of the KNN-PCA Cloud and the corresponding schemas within that region. Doing so identifies the maximal subschema contained in all subschemas and regenerates the schema summary pane based only on the segment containing the maximal subschema. The maximal subschema itself is also highlighted in the schema summary pane. On the Covariance cloud, the lasso tool behaves similarly, selecting attributes explicitly rather than a maximal subschema.

The reverse mapping is achieved using highlighting, as illustrated in Figure 4.7. Users can select one or more attributes (or groupings) in the schema summary pane, and the KNN-PCA Cloud (resp., Covariance Cloud) is modified to highlight schemas in the corresponding segment (resp., to highlight the attributes). In conjunction with their prior knowledge of their tasks and ideal use cases, we use this approach to perform the initial schema segmentation.

In either case, after selecting a set of attributes or schemas, the analyst may choose to drill down into the selected segment, regenerating both views for the now restricted collection of schemas.

4.3.3 An Iterative Approach

At any time in our exploration pipeline analysts may stop where they are, take the knowledge they gained about their dataset, and restart the process from the beginning. Through exploring the Twitter dataset we found retweets and quoted tweets to have a high correlation, with this knowledge we can then start back at the beginning and depending on whether our task allows us to merge these attributes,

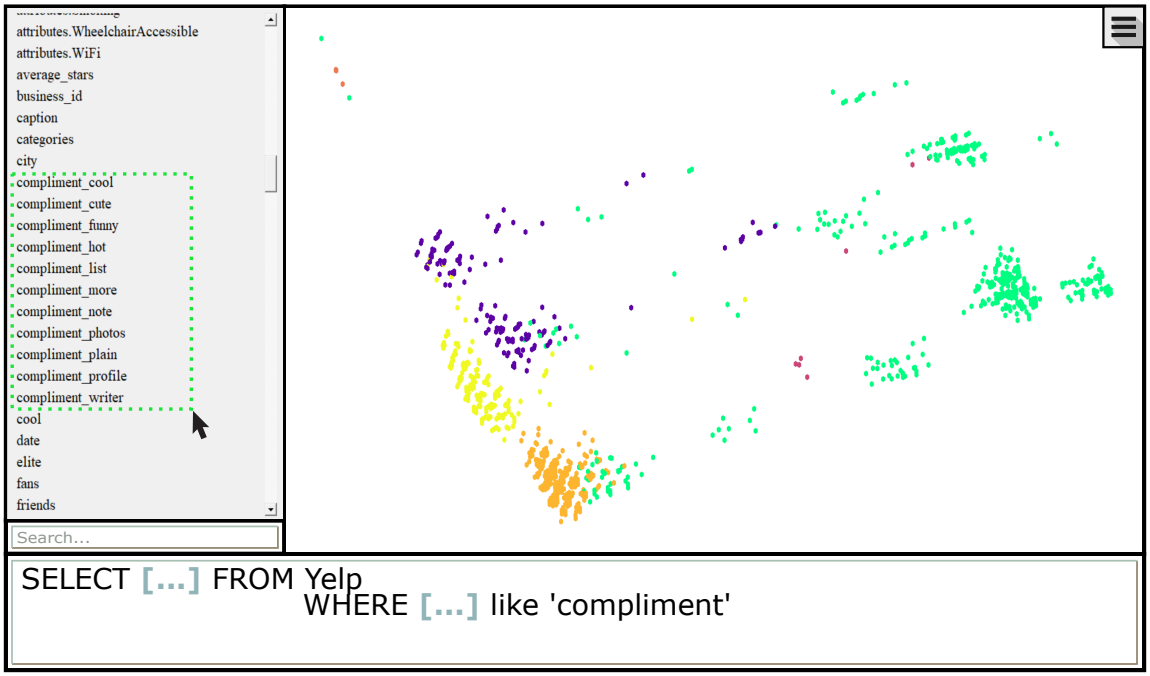


Figure 4.7: Points containing user selected attributes are highlighted green.

we can then choose a more appropriate K value for KNN. In addition, attributes that have no relationship to core attributes, such as a users profile_image_url being present, can easily be pruned to compress our summary output. By incrementally learning about their dataset, an analyst can converge on the views required for their tasks.

4.4 Related Work

Schema Extraction. Schema extraction for Json, as well as for other self-describing data models like XML has seen active interest from a number of sources. An early effort to summarize self-describing *hierarchical* data can be found in the LORE system's DataGuides [**?**]. DataGuides view schemas begin with a forest of tree-shaped schemas and progressively merge schemas, deriving a compact encoding of the forest

as a DAG. Although initially designed for XML data, similar ideas have been more recently applied for Json data as well [**?**, **?**]. Key challenges in this space involve simply extracting schemas from large, multi-terabyte collections of Json data [**?**], as well as managing ambiguity in the set of possible factorizations of a schema [**?**, **?**]. The approach taken by Baazizi et. al. [**?**] in particular adopts a type unification model similar to ours, but lacks the conjunctive operator of our type-system. For non-hierarchical data, interactive tools like Wrangler [**?**] provide an interactive frameworks for regularizing schemas.

Physical Layout. While schemas play a role in the interpretability of a Json data set, they can also help improve the performance of Json queries. One approach relies on inverted indexes [**?**] to quickly identify records that make use of sparse paths in the schema. Another approach is to normalize schema elements [**?**]. Although the resulting schema may not always be interpretable, this approach can result in substantial space savings.

Information Retrieval. From a more general perspective, the schema extraction problem which aims at making large datasets tractable for interactive exploration, is an instance of *categorization* problem that has been repeatedly studied in the literature. More precisely, attributes (metadata) of the datasets can be grouped into a hierarchy of "facets" (i.e., categories) [**?**] where the child-level facets are conditioned on the presence of the parent one. In our approach, we adopt the hierachical data visualization and focus more on the algorithmic essence of the problem: How to (1) balance between the preciseness and conciseness of the visualization and (2) respond to users' data exploration requests in a scalable way.

4.5 Future Work

One challenge that we will need to address in SchemaDrill is coping with nested collections. At the moment, the user can manually merge collections of attributes that correspond to disjoint entites. However, we would like to automate this process. One observation is that a typical collection like an array has a schema with the general structure:

$$(P_1 \vee P_1P_2 \vee P_1P_2P_3 \vee \ldots) \;=\; (P_1 \wedge (\emptyset \vee P_2 \wedge (\emptyset \vee P_3 \wedge (\ldots))))$$

The version of this expression on the right hand side is notable as its closure over the semiring $\left\langle \; \{\{\mathbf{P}\}\}, \vee, \wedge, \emptyset, \{\{\}\} \; \right\rangle$ would indicate that the semiring is "quasiregular" or "closed", an algebraic structure best associated with the Kleene star. Hence, we plan to explore the use of the Kleene star to encode nested collections in our algebra. A key challenge in doing so is detecting opportunities for incorporating it into a summary, a more challenging form of the factorization problem.

A further step to increase the capabilities of SchemaDrill is to incorporate type information in the summarization. This adds an extra layer of information an analyst can extract from our system, as well as the ability to identify and correct schema errors. As a long term goal we will provide capabilities for linking views, for example by defining functional dependencies. The goal is to create full entity relationship diagrams. In particular, one interesting way to identify potential relationships that exist between entities is by leveraging the overlap between segments.

Chapter 5

Application: Summarizing Probabilistic Databases

5.1 Introduction

Many tasks in machine learning, information extraction, etc, generate uncertain data. That is, output tuples generated from one probabilistic prediction differ from the other. Though it is a general practice to deterministically output the most probable prediction, it is beneficial to explore other possible predictions, especially when an application needs to compare predictions under different parameter tuning strategies [**?**]. However, as [**?**] pointed out, state-of-the-art statistical models of information extraction provide a sound probability distribution over extractions but are challenging to represent and query in relational framework.

One naive strategy would be letting the application to iterate over all observed outputs, together with their probabilities of being observed. This strategy, on one extreme, defines a *probabilistic database* [**?**] and walks the application through the *entire space* of possible worlds. This is a sufficient condition for the application to

discover all information that may be useful: the probability that any set of tuples will be contained in the output. However, this naive strategy puts an unnecessary burden on the application side. That is, there are practically exponential number of possible outputs [**?**]. Each observed output may contain duplicate tuples and it is possible to deliver the probability distribution over possible outputs in a more compact way.

The other naive strategy would be letting the application view each *individual* tuple together with the probability that the tuple is contained in an output. The probability of any *set* of tuples can then be estimated by multiplying the probability of each content tuple. This strategy, on the other extreme, defines a *tuple-independent probabilistic database* [**?**] that compactly represents the distribution of outputs by assuming each tuple independently exists in an output. However, for applications where tuples are predominantly correlated in the output, probabilities delivered by tuple-independent probabilistic database may greatly deviate from the truth [**?**]. Consider an example application where tuple-correlation is dominant by nature.

Example 10. *Consider an* Enitity Resolution (ER) *task where the input is a set of tuples where each tuple contains a set of attributes that may help to determine whether two tuples are different facets of the same entity. The output will be the same set of tuples but augmented with an additional attribute indicating its predicted entity ID.*

Since the prediction is uncertain, the output data can be viewed as a space of possible sets of augmented tuples that a predictor may generate. Augmented tuples in the output are correlated by design, that is, given that a tuple $\mathbf{t}$ is assigned entity ID *eid*, other tuples having attributes similar to $\mathbf{t}$ being assigned the same *eid* increases and are no longer independent events. This example can be generalized to

classification or clustering related tasks where the output is a set of label-augmented tuples where the label assignments are uncertain.

To encode tuple correlation, various kind of auxiliary information (e.g., *c-tables* [**?**], factors [**?**, **?**, **?**], Markov Logic [**?**]) are introduced for delivering more accurate probabilities. Auxiliary information inevitably increases the model complexity, making the inference of the desired probabilities harder. It also puts a heavier burden on the application side for digesting the representation.

To carefully introduce auxiliary information, we start by considering *c-tables* [**?**]. A c-tables compactly represents the distribution over outputs by presenting only individual tuples, annotated with a propositional formula. A tuple exists in the output iff its propositional formula evaluates to be true. By walking the application through a c-table as well as the corresponding probability distribution over variables in its formula, we equivalently delivers the distribution of possible outputs. However, linking the probability distribution with its c-table generally requires heavy weight inference. That is, to compute the probability of a set of tuples being co-existent, we need to enumerate all possible evaluations of their variables such that their propositional formula are true. This process is non-trivial especially when we need to repeated compute such probabilities for different sets of tuples. Moreover, in our example application, it is generally difficult to obtain such propositional formula, since tuple correlation is caused by the complex nature of some predictor. As a result, for our target applications, we only expect a tuple in the c-table to be annotated with a single distinct variable X_i with $X_i = 1$ meaning the presence of the ith tuple in the output. The probability of two tuples $\mathbf{t}_i, \mathbf{t}_j$ being co-existent is directly linked with the marginal probability $p(X_i = 1, X_j = 1)$. To deliver the distribution of outputs, we need to walk an application through the c-table and the joint distribution $\mathbf{P} = p(\ldots, X_i, \ldots)$. Enumerating all valuations of the variables is not practical.

Instead, we group correlated variables together and only enumerate valuation within each group. This essentially creates a *factor* for each group. Mappings in a factor is still exponential with respect to the number of correlated variables. In this paper, we study the problem of encoding the joint distribution $\mathbf{P}$ by only a subset of mappings in factors. We show how desired probabilities of the distribution $\mathbf{P}$ can be computed by the lossy encoding. Due to incomplete mappings, the computed probabilities generally deviates from the truth. However, the encoding allows finer granularity in trading-off between conciseness and probability computation accuracy. That is, by gradually populating the mappings in a factor, an application can choose to obtain an encoding that incurs a high-fidelity in computing probabilities but is verbose, or obtain a more compact encoding that incurs a greater loss of accuracy. To manage the loss-rate, we develop a framework for reasoning about the trade-off between verbosity and fidelity. While the encoding does not admit closed-form solutions to classical information theoretical fidelity measures like information loss, we propose an efficiently computable fidelity measure called Reproduction Error.

Contributions.

Roadmap.

5.2 Problem Definition

Input Definition. As shown in Figure 5.1, the input data consists of a finite set of possible instances $\mathbf{D} = \{D_1, \ldots, D_n\}$ where each instance D_i consists of a set of tuples. We use the *possible worlds semantics* [**?**] and call $\mathbf{D}$ as *possible worlds*. Each world is assigned a probability by the distribution $\mathbf{P} : \mathbf{D} \rightarrow [0, 1]$ such that $\sum_i \mathbf{P}(D_i) = 1$. In other words, our input is defined as a discrete probability space $PDB = (\mathbf{D}, \mathbf{P})$.

Goal. An application theoretically may require scanning through all possible worlds, in order to aggregate the probabilities where the corresponding worlds contain some target *set* of tuples. In the context of Entity Resolution task, an application may require aggregating probabilities of all predictions that contain tuples labeled by a target *eid*. For convenience, we will refer to these aggregated probabilities as *marginal probabilities* or simply *marginals* in the rest of the paper. Our goal is then to help the application efficiently explore and compute marginals in a practical way without iterating through a large number of possible worlds.

System Overview. We begin by giving the reader an overview on how the input flows *theoretically* and *practically* to a client-side application. In Figure 5.1, the red dashed lines encompasses a theoretical data flow that iterates through all possible worlds. In practice, possible worlds in the input will converted into an initial form of representation. Based on the initial form, further compression can be made that provides a materialized view over the input. The materialized view can be efficiently queried and the query result will be simplified before being presented to the application.

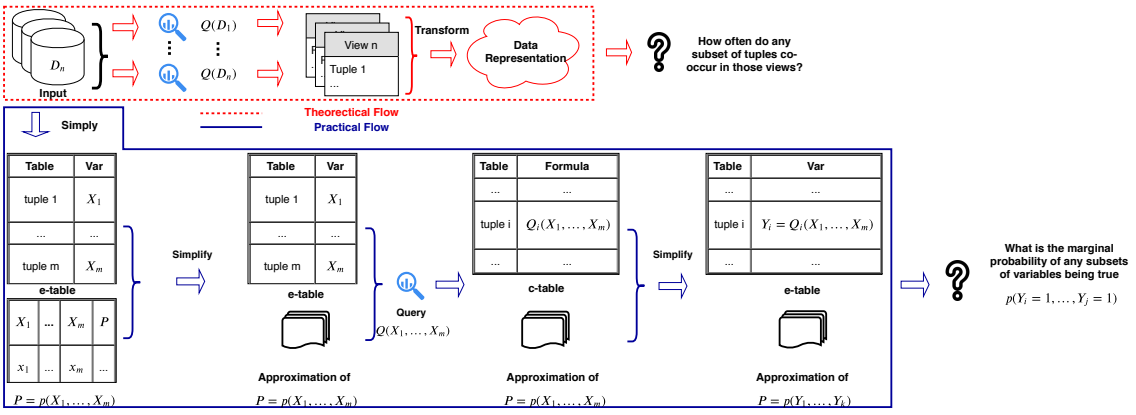


Figure 5.1: **Theoretical v.s. Practical Data-flows.**

5.2.1 Practical Input Representation

The previous definition of **D** does not suggest a practical representation, especially when explicit enumeration of all possible worlds is not feasible when n is very large.

C-tables and pc-tables. We adopt *c-tables* to compactly represent **D**. A *c-table* [**?**] is a set of *distinct* tuples contained in all possible worlds, with each tuple annotated by a propositional formula over random variables. A *valuation* over these variables assigns each propositional formula a truth value, indicating the existence of the corresponding tuple in a world. There is a many-to-one mapping from valuations to a possible world. Extending a c-table by a joint probability distribution over random variables, the resulting *pc-table* can represent any probabilistic database PDB [**?**].

Simplified C-tables. For our example application, there are two limitations by adopting a general c-table representation. Firstly, it is difficult to obtain such propositional formula due to lack of knowledge on how tuples are correlated in PDB. That is, due to complex nature of the function that generates PDB, it is generally hard to formulate how tuples are correlated. Secondly, the mapping from valuations of variables to possible worlds is many-to-one for a general c-table. To compute desired marginals, it may require an expensive aggregate over all valuations that map to the same world.

To overcome the limitations, we represent our input by a family of simplified c-tables, which we call *existence tables* or simply *e-tables*. Each tuple $\mathbf{t}_i$ in an e-table is annotated with a single random variable X_i whose value is 1, meaning that the ith tuple exists in a world, or 0 otherwise. We refer to these variables as *existence variables*. There is an *one-to-one* mapping from a valuation of existence variables $(x_1, \ldots, x_n) \in \{0, 1\}^n$ to a world D. The probability of the world (i.e., $\mathbf{P}(D)$) can

be computed directly from the joint probability distribution $p(X_1 = x_1, \ldots, X_n = x_n)$. When it is clear from context, we abuse notation and denote the distribution $p(X_1 = x_1, \ldots, X_n = x_n)$ also as $\mathbf{P}$. We call an e-table, together with $\mathbf{P}$, a *pe-table*.

5.2.2 Summarizing Joint Distribution

It is not practical, both in space and marginal computation efficiency, to present pe-table in a view that iterates through all valuations (i.e., worlds) of existence variables. Hence we need a view that summarizes $\mathbf{P}$ into a more compact form.

Naive Summary. There can be a set of tuples $\{\ldots, \mathbf{t}_i, \ldots\}$ independent of each other such that,

$$p(\ldots, X_i = 1, \ldots) = \Pi_i \; p(X_i = 1)$$

In other words, we do not need to store valuations of any combination of those variables in the view, which greatly reduces the number of parameters. We name the set of parameters $p(X_i = 1)$ as a *naive summary* and return to it through out the rest of this paper. The naive summary is often referred to as *tuple-independent* model [**?**]. When variables are correlated, which is prevalent in our example application, the marginals computed from a naive summary are error-prone.

Loss-less Summary. For tuples whose variables that are correlated, a *factor graph* [**?**] that is equivalent to the marginal probability distribution over these variables, is a sufficient condition for computing any relevant marginals. However, it requires heavy weight inference to obtain such set of factors and for each factor, the number of parameters is still exponential to the number of variables that the factor contains.

Lossy Summary. Practically an application may not require computing exact marginals for *every* set of tuples and we can *cache* marginals that are frequently

computed and offer estimation on others. Specifically, in our example Entity Resolution task, an application may require to explore labelled tuples that are contained in *all* possible predictions, and tuples that are (in-)frequently or otherwise statistically significant. The application may tolerate inexact marginal computation for other sets of tuples. This observation motivates us to study the option of a *lossy* summary containing only a limited number of marginals and offers estimation on marginals not contained.

5.2.3 Pattern-based Summary

Patterns. We name a set of co-existing tuples as a *pattern*. More formally we define a *pattern* as an arbitrary binary vector $\omega = (x_1, \ldots, x_n)$ with $x_i = 1$ meaning the ith tuple being present in the pattern. A pattern ω is *contained* in the other $\bar{\omega}$ if $\forall i, x_i \leq \bar{x}_i$, denoted as $\omega \subseteq \bar{\omega}$. Note that a world D is a pattern by definition. Randomly drawing a world $D \in \mathbf{D}$ from PDB, the probability that it contains pattern ω is defined by:

$$p(\mathbf{D} \supseteq \omega) = \sum_i \mathbf{P}(I_i)\mathbf{1}_\omega(I_i) \text{ where } \mathbf{1}_\omega(I_i) = \begin{cases} 1 & \text{if } I_i \supseteq \omega \\ 0 & \text{otherwise} \end{cases}$$

In pe-tables, $p(\mathbf{D} \supseteq \omega)$ is equivalent to marginal probability $p(X_1 \geq x_1, \ldots, X_n \geq x_n)$. We call $p(\mathbf{D} \supseteq \omega)$ as the *marginal* of a pattern.

Pattern-based Summary. Denote the mapping $\mathcal{E}_{max}$ from each pattern ω to its marginal:

$$\mathcal{E}_{max} = \{\ \omega \rightarrow p(\mathbf{D} \supseteq \omega) \mid \omega \in \{0,1\}^n\ \}$$

A *pattern-based summary* $\mathcal{E}$ is any such partial mapping $\mathcal{E} \subseteq \mathcal{E}_{max}$. We denote

the marginal of pattern ω in summary $\mathcal{E}$ by $\mathcal{E}[\omega]$ ($= p(\mathbf{D} \supseteq \omega)$). When it is clear from context, we abuse syntax and also use $\mathcal{E}$ to denote the set of patterns it maps (i.e., $domain(\mathcal{E})$). Hence, $|\mathcal{E}|$ is the number of mapped patterns, which we call the summary's *verbosity*.

5.3 Marginal Estimation

A pattern-based summary not only delivers marginals that it contains, but also helps in estimating those that are not explicitly given.

Naive Summary Revisited. Before detailed explanation, we first consider an intuitive example. There is one specific family of pattern-based summaries that treat each tuple as being independent, which is equivalent to naive summaries discuss in Section 5.2.1. As a re-phrase, we call a pattern-based summary as a naive summary $\ddot{\mathcal{E}}$ if it is composed of all patterns that have exactly one possible tuple $\mathbf{t}_i$ with marginal $p(X_i \geq 1)$:

$$domain(\ddot{\mathcal{E}}) = \{\ (0, \ldots, 0, x_i, 0, \ldots, 0) \mid\ i \in [1, n],\ x_i = 1\ \}$$

To estimate marginals not given in the summary, we apply multiplication

$$p(X_1 \geq x_1, \ldots, X_n \geq x_n) = \Pi_{i=1,n}\, p(X_i \geq x_i)$$

Patterns as Constraints. For an arbitrary summary $\mathcal{E}$, we characterize its process of marginal computation by defining a *space* (denoted by $\Omega_{\mathcal{E}}$) of distributions $\rho \in \Omega_{\mathcal{E}}$ allowed by a summary $\mathcal{E}$. This space is defined by constraints as follows: First, we have the general properties of probability distributions:

$$\forall\, D_i \in \{0,1\}^n : \rho(D_i) \geq 0 \qquad \sum_{i=1,2^n} \rho(D_i) = 1$$

Each pattern ω in the summary $\mathcal{E}$ constrains relevant probabilities in distribution ρ to sum to the target marginal:

$$\forall \omega \in \mathit{domain}(\mathcal{E}) : \;\; \mathcal{E}[\omega] = \sum\nolimits_{D \sqsupseteq \omega} \rho(D)$$

The dual constraints $1 - \mathcal{E}[\omega] = \sum_{D \not\sqsupseteq \omega} \rho(D)$ are redundant under the constraint $\sum_i \rho(D_i) = 1$.

The resulting space $\Omega_\mathcal{E}$ is the set of all permitted distributions, or equivalently all input PDB, that obey these constraints. As a result, the distribution $\rho \in \Omega_\mathcal{E}$ that the summary delivers is ambiguous: We model this ambiguity using a random variable $\mathcal{P}_\mathcal{E}$ with support $\Omega_\mathcal{E}$. The true distribution $\mathbf{P}$ must appear in $\Omega_\mathcal{E}$, denoted as $\mathbf{P} \equiv \rho^* \in \Omega_\mathcal{E}$ (i.e., $p(\mathcal{P}_\mathcal{E} = \rho^*) > 0$). Of the remaining distributions ρ admitted by $\Omega_\mathcal{E}$, it is possible that some are more likely than others. This prior knowledge may be modeled as a prior on the distribution of $\mathcal{P}_\mathcal{E}$ or equivalently by an additional constraint. However, for the purposes of this paper, we take the uninformed prior by assuming that $\mathcal{P}_\mathcal{E}$ is uniformly distributed over $\Omega_\mathcal{E}$:

$$p(\mathcal{P}_\mathcal{E} = \rho) = \begin{cases} \frac{1}{|\Omega_\mathcal{E}|} & \text{if } \rho \in \Omega_\mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

The summary delivers marginals by randomly picking one permitted distribution from the entire space and aggregate probabilities accordingly.

5.4 Summary Accuracy

The accuracy of such delivery can be considered from two related, but subtly distinct perspectives: (1) *Ambiguity* measures how much room the summary leaves for interpretation and (2) *Deviation* measures how reliably the summary approximates the target distribution ρ^*.

Ambiguity. We define the Ambiguity $\Gamma(\mathcal{E})$ of a summary as the entropy of the random variable $\mathcal{P}_\mathcal{E}$. The higher the entropy, the less precisely $\mathcal{E}$ identifies a specific distribution.

$$\Gamma(\mathcal{E}) = \sum_{\rho} p(\mathcal{P}_\mathcal{E} = \rho) \log \left(p(\mathcal{P}_\mathcal{E} = \rho) \right)$$

Deviation. The deviation from any permitted distribution ρ to the true distribution ρ^* can be measured by the Kullback-Leibler (K-L) divergence $\mathcal{D}_{KL}(\rho^* || \rho)$. We define the Deviation $\mathbb{D}(\mathcal{E})$ of a summary as the expectation of the K-L divergence over all permitted $\rho \in \Omega_\mathcal{E}$:

$$\mathbb{D}(\mathcal{E}) = \mathbb{E}_{\mathcal{P}_\mathcal{E}} \left[\mathcal{D}_{KL}(\rho^* || \mathcal{P}_\mathcal{E}) \right] = \sum_{\rho \in \Omega_\mathcal{E}} p(\mathcal{P}_\mathcal{E} = \rho) \cdot \mathcal{D}_{KL}(\rho^* || \rho)$$

Limitations. Neither Deviation nor Ambiguity has a closed-form formula and it is not practical to enumerate the infinitely large space. Authors in [**?**] suggests that a single *representative* can be chosen from the space that approximates true distribution ρ^*.

Maximum Entropy Distribution. The representative distribution is chosen by applying the maximum entropy principle [21] commonly used in pattern-based

encoders [15, 16]. That is, we select the distribution $\overline{\rho}_{\mathcal{E}}$ with maximum entropy:

$$\overline{\rho}_{\mathcal{E}} = \arg\max_{\rho \in \Omega_{\mathcal{E}}} \mathcal{H}(\rho) \qquad \text{where } \mathcal{H}(\rho) = \sum_{i=[1,2^n]} -\rho(I_i) \log \rho(I_i)$$

The maximum entropy distribution best represents the current state of knowledge. That is, a distribution with lower entropy assumes additional constraints derived from patterns that we do not know, while one with higher entropy violates the constraints from patterns we do know.

Maximizing an objective function belonging to the exponential family (entropy in our case) under a mixture of linear equality/inequality constraints is a convex optimization problem [22] which guarantees a *unique* solution and can be solved using the cvx toolkit [23], and/or by *iterative scaling* [15, 16].

Naive Summary revisited. For naive summaries $\ddot{\mathcal{E}}$ specifically, $\overline{\rho}_{\ddot{\mathcal{E}}}$ has the closed-form representation:

$$\overline{\rho}_{\ddot{\mathcal{E}}}(I) = \prod_i p(X_i = x_i) \qquad \text{where } I = (x_1, \ldots, x_n)$$

Reproduction Error. We define *Reproduction Error* $e(\mathcal{E})$ of summary $\mathcal{E}$ as the entropy difference between the representative and true distribution:

$$e(\mathcal{E}) = \mathcal{H}(\overline{\rho}_{\mathcal{E}}) - \mathcal{H}(\rho^*) \qquad \text{where } \overline{\rho}_{\mathcal{E}} = \arg\min_{\rho \in \Omega_{\mathcal{E}}} -\mathcal{H}(\rho)$$

It has been proved [**?**] that Reproduction Error closely parallels both Ambiguity and Deviation.

5.4.1 Query Accuracy

We then study the accuracy of pattern-based summaries when the corresponding probabilistic databases are involved in three basic relational operations: (1) *join*, (2) *selection* and (3) *project*. Specifically, we study how to modify existing pattern-based summaries in order to suit these operations and give upper-bounds on Reproduction Error of the resulting summaries.

5.4.1.1 Accuracy under join

Consider two uncertain tables, or equivalently possible worlds $\mathbf{P}_1, \mathbf{P}_2$ under *join* operation. Recall that possible worlds can be translated into joint distribution, that is, we have $\mathbf{P}_1 = p(\ldots, X_i, \ldots)$ and $\mathbf{P}_2 = p(\ldots, Y_j, \ldots)$ where X_i, Y_j indicate the presence of i, jth tuple in possible worlds $\mathbf{P}_1, \mathbf{P}_2$ respectively. The resulting uncertain table after *join*, or equivalently possible worlds $\mathbf{P}_{1\times 2} = \mathbf{P}_1 \times \mathbf{P}_2$ can also be translated into joint distribution $\mathbf{P}_{1\times 2} = p(\ldots, Z_{i,j}, \ldots)$ where $Z_{i,j} = 1$ iff $X_i = 1$ and $Y_j = 1$.

Join-derived summary. Consider pattern-based summaries $\mathcal{E}_1, \mathcal{E}_2$ for $\mathbf{P}_1, \mathbf{P}_2$ with corresponding maximum entropy distribution $\overline{\rho}_{\mathcal{E}_1}, \overline{\rho}_{\mathcal{E}_2}$. The resulting summary $\mathcal{E}_{1\times 2}$ can be obtained from *joining* patterns from both summaries and multiplying their corresponding marginals. Denote the resulting maximum entropy distribution after *join* as $\overline{\rho}_{\mathcal{E}_{1\times 2}}$.

Upper-bound of Reproduction Error. By definition, the Reproduction Error $e(\mathcal{E}_{1\times 2})$ of summary $\mathcal{E}_{1\times 2}$ measures the entropy difference $\mathcal{H}(\overline{\rho}_{\mathcal{E}_{1\times 2}}) - \mathcal{H}(\mathbf{P}_{1\times 2})$. We would like to approximate the two entropies to avoid populating mappings in distributions $\overline{\rho}_{\mathcal{E}_{1\times 2}}$ and $\mathbf{P}_{1\times 2}$. That is, we would like to provide an upper-bound $\tilde{e}(\mathcal{E}_{1\times 2})$ and such that $e(\mathcal{E}_{1\times 2}) \leq \tilde{e}(\mathcal{E}_{1\times 2})$.

Observe that there is a many-to-one mapping from valuations of jointly distributed variables $\mathbf{P}_{1,2} = p(\ldots, X_i, \ldots, Y_j, \ldots)$ to $\mathbf{P}_{1\times 2}$. That is, by fixing the valuation of all other variables except for some pair X_i, Y_j, all three cases where $X_i = 0 \wedge Y_j = 0$, $X_i = 0 \wedge Y_j = 1$ or $X_i = 1 \wedge Y_j = 0$ map to the same valuation of $\mathbf{P}_{1\times 2}$ where $Z_{i,j} = 0$. Hence, the entropy $\mathcal{H}(\mathbf{P}_{1\times 2}) \leq \mathcal{H}(\mathbf{P}_{1,2})$. Trivially we also have $\mathcal{H}(\mathbf{P}_{1\times 2}) \geq \max(\mathcal{H}(\mathbf{P}_1), \mathcal{H}(\mathbf{P}_2))$. Since X_i are independent of Y_j in $\mathbf{P}_{1,2}$, we have $\mathcal{H}(\mathbf{P}_{1,2}) = \mathcal{H}(\mathbf{P}_1) + \mathcal{H}(\mathbf{P}_2)$. We have similar observation for maximum entropy distributions: $\mathcal{H}(\overline{\rho}_{\mathcal{E}_{1\times 2}}) \leq \mathcal{H}(\overline{\rho}_{\mathcal{E}_1}) + \mathcal{H}(\overline{\rho}_{\mathcal{E}_2})$. We thus compute the upper-bound $\tilde{e}(\mathcal{E}_{1\times 2}) = \mathcal{H}(\overline{\rho}_{\mathcal{E}_1}) + \mathcal{H}(\overline{\rho}_{\mathcal{E}_2}) - \max(\mathcal{H}(\mathbf{P}_1), \mathcal{H}(\mathbf{P}_2))$ or a looser but simpler bound $\mathcal{H}(\overline{\rho}_{\mathcal{E}_1}) + \mathcal{H}(\overline{\rho}_{\mathcal{E}_2})$.

5.4.1.2 Accuracy under selection predicate

Consider the case where only a subset of tuples in possible worlds $\mathbf{P} = p(\ldots, X_i, \ldots)$ are chosen, according to some arbitrary *selection* predicate σ. For convenience, we abuse notation and also represent the indices of its chosen tuples as σ. The resulting possible worlds, or equivalently the joint distribution $\mathbf{P}_\sigma$ is the marginal distribution $p(\ldots, X_{i\in\sigma}, \ldots)$ by summing out variables $X_{i\notin\sigma}$.

Selection-derived summary. Consider a pattern-based summary $\mathcal{E}$ for $\mathbf{P}$, the resulting summary $\mathcal{E}_\sigma$ for $\mathbf{P}_\sigma$ can be constructed in three steps: (1) remove tuples that are not chosen by σ for each pattern in $\mathcal{E}$, (2) merge patterns that are the same and add their marginals and (3) remove empty patterns.

Upper-bound of Reproduction Error. Suppose we encode $\mathbf{P}$ first by naive summary $\ddot{\mathcal{E}}$ whose maximum entropy distribution $\overline{\rho}_{\ddot{\mathcal{E}}}$ is equivalent to tuple-independent model. The resulting naive summary $\ddot{\mathcal{E}}_\sigma \subseteq \ddot{\mathcal{E}}$ after *select* operation is a subset of single-tuple patterns filtered by the predicate σ. Reproduction Error of the naive

summary $\ddot{\mathcal{E}}_\sigma$ is thus computed as $e(\ddot{\mathcal{E}}_\sigma) = \sum_{i\in\sigma} \mathcal{H}\big(p(X_i)\big) - \mathcal{H}(\mathbf{P}_\sigma)$. In cases when we do not want to populate mappings of distribution $\mathbf{P}_\sigma$ for each given predicate, we can simply compute $\tilde{e}(\ddot{\mathcal{E}}_\sigma) = \sum_{i\in\sigma} \mathcal{H}\big(p(X_i)\big)$ which is the upper-bound of $e(\ddot{\mathcal{E}}_\sigma)$ for all possible σ. For any other summary $\mathcal{E}_\sigma$ that contains naive summary $\mathcal{E}_\sigma \supseteq \ddot{\mathcal{E}}_\sigma$, it may improve naive summary by marginals that encode tuple-correlation. Hence the Reproduction Error $e(\mathcal{E}_\sigma)$ is also upper-bounded by $\tilde{e}(\ddot{\mathcal{E}}_\sigma)$. Trivially, we also have $e(\mathcal{E}_\sigma) \leq \mathcal{H}(\overline{\rho}_\mathcal{E})$. Hence the Reproduction Error of any summary $\mathcal{E}_\sigma \supseteq \ddot{\mathcal{E}}_\sigma$ is upper-bounded by $\min\big(\tilde{e}(\ddot{\mathcal{E}}_\sigma), \mathcal{H}(\overline{\rho}_\mathcal{E})\big)$.

5.4.1.3 Accuracy under project

Consider the case where tuples in uncertain table $\mathbf{P}$ are *projected* onto an arbitrary set of attributes A. We group tuples whose values are the same on projected attributes A and create a new random variable Z for each group, where $Z = 0$ iff $X_i = 0$ for all tuples $\mathbf{t}_i$ in the group, otherwise $Z = 1$. The possible worlds distribution $\mathbf{P}_{\pi_A}$ after *project* operation, is thus the joint distribution $p(Z_1, \ldots, Z_k)$ where k is the total number of tuple groups.

Project-derived summary. Suppose we have summary $\mathcal{E}$ for $\mathbf{P}$. The resulting summary $\mathcal{E}_{\pi_A}$ for $\mathbf{P}_{\pi_A}$ can be constructed by (1) applying *project* on each pattern and (2) merge patterns that are the same after *project* and add their marginals accordingly.

Upper-bound of Reproduction Error. We fix the valuations for all Z_i except for the kth group Z_k. Equivalently we fix all valuations of X_i in $\mathbf{P}$ except for those X_j that belong to kth group. Then we notice that there is a many-to-one mapping from all valuations of X_j in the group, where at least one $X_j = 1$, to $Z_k = 1$. In other words, we have $\mathcal{H}(\mathbf{P}_{\pi_A}) \leq \mathcal{H}(\mathbf{P})$. Similarly we also have $\mathcal{H}(\overline{\rho}_{\mathcal{E}_{\pi_A}}) \leq$

$\mathcal{H}(\overline{\rho}_{\mathcal{E}})$ for maximum entropy distributions of corresponding summaries. That is, the Reproduction Error of summary $\mathcal{E}_{\pi_A}$ for any *project* operation π_A is upper-bounded by $\mathcal{H}(\overline{\rho}_{\mathcal{E}}) - \mathcal{H}(\mathbf{P}_{\pi_A})$ or a looser but simpler bound $\mathcal{H}(\overline{\rho}_{\mathcal{E}})$.

5.5 Constructing Summaries

5.6 Related Work

In this section, we walk through related work on encoding tuple correlation in probabilistic databases using various kinds of auxiliary information.

C-tables. The canonical representation of probabilistic databases is *c-table* [**?**]. A *c-table* is a table with each tuple annotated with an additional attribute: a propositional formula over random variables. Two tuples are correlated if random variables in their propositional formula depends on each other. Given a query over a c-table, the correlation among tuples in the query answer is encoded by the *lineage* (or provenance [**?**]) of each tuple, which is also a propositional formula.

Based on c-tables, Trio system [**?**] resolves tuple correlation by tracking lineage of tuples in query processing. By defining a probability space over the assignments of the random variables in lineage of tuples, a *confidence value* can be computed for each tuple in the query answer. The probability of some answer to be chosen as the "correct" one is proportional to the product of confidence values of all of its tuples. Similar idea can be found in MayBMS [**?**]. However, as [**?**] pointed out, there are cases where lineage is not obtainable. In our example application, tuples are correlated in a latent, complex way that populating propositional formula for tuples in the c-table is difficult.

Graphical models and Factors. Enlightened by Probablistic Relational Model

(PRM) [?], numerous works [?, ?] use *probabilistic graphical models* (PGM) to encode tuple correlation directly by probabilities of co-existing tuples. More specifically, considering any set of co-existing tuples, encoded as a binary vector $\mathbf{v} = (x_1, \ldots, x_n)$ with $x_i = 1$ indicating the presence of the ith possible tuple in the co-existence relationship. A *factor function* or simply *factor* $f : \mathbf{v} \in \{0, 1\}^n \rightarrow [0, 1]$ maps any set of co-existing tuples $\mathbf{v}$ to its probability of being present in the probabilistic database. Since there are exponentially large (i.e., 2^n) number of mappings, instead of having a single factor that takes n variables as input, we can have multiple factors, each takes only a small subset of variables that are correlated. If two factors do not share variables, the variables in one are considered independent of those in the other. By multiplying the outputs of the two factors, we can efficiently compute the probability of any combination of variables in two factors.

Markov Logic Networks. Jha and Suciu [?] shows how to automatically generate factors and populate their content mappings using prior knowledge on tuple-correlation, expressed by the language of First Order Logic. Similar technique is implemented in DeepDive system [?].

As authors of MCDB [?] pointed out, most previously discussed approaches augment data with attribute-level or tuple-level annotations, which are loaded into the database along with the data itself. The mixture of data and uncertainty annotation is inflexible when changes have to be made to the underlying uncertainty model (e.g., lineage), as all related annotations typically need to be recomputed outside of the database and then loaded back in. To facilitate maintenance, MCDB proposes to separate uncertainty model from the data.

Monte Carlo approach. More specifically, MCDB allows a user to define arbitrary *variable generation* (VG) functions that encodes the uncertainty model. That

is, treating a VG function as a black box, MCDB use it to pseudo-randomly generate samples of possible instances on the fly and run queries over the samples. Interestingly, sample instances are sets of co-existing tuples. By assigning a probability to each sample, the probability of any co-existing tuples that are contained in any sample is estimated naturally by iterating through the samples.

Bibliography

[1] Bikash Chandra, Bhupesh Chawda, B. Kar, K. V. Reddy, Shetal Shah, and S. Sudarshan. Data generation for testing and grading SQL queries. *VLDBj*, 2015.

[2] Nicolas Bruno and Surajit Chaudhuri. Automatic physical database tuning: A relaxation-based approach. In Fatma Özcan, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 227–238. ACM, 2005.

[3] Cynthia Dwork. Differential privacy. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2006.

[4] Gloria Chatzopoulou, Magdalini Eirinaki, Suju Koshy, Sarika Mittal, Neoklis Polyzotis, and Jothi Swarubini Vindhiya Varman. The querie system for personalized query recommendations. *IEEE Data Eng. Bull.*, 34(2):55–60, 2011.

[5] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C. Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. Self-driving database management systems. In *CIDR*, 2017.

[6] Gökhan Kul, Duc Luong, Ting Xie, Patrick Coonan, Varun Chandola, Oliver Kennedy, and Shambhu J. Upadhyaya. Ettu: Analyzing query intents in corporate databases. In Jacqueline Bourdeau, Jim Hendler, Roger Nkambou, Ian Horrocks, and Ben Y. Zhao, editors, *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11-15, 2016, Companion Volume*, pages 463–466. ACM, 2016.

[7] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Information Theory*, 23(3):337–343, 1977.

[8] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Information Theory*, 24(5):530–536, 1978.

[9] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, Sept 1952.

[10] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788, 1999.

[11] Ian T. Jolliffe. Principal component analysis. In Miodrag Lovric, editor, *International Encyclopedia of Statistical Science*, pages 1094–1096. Springer, 2011.

[12] David M. Blei. Probabilistic topic models. *Commun. ACM*, 55(4):77–84, 2012.

[13] Dingding Wang, Shenghuo Zhu, Tao Li, and Yihong Gong. Multi-document summarization using sentence-based topic models. In *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore, Short Papers*, pages 297–300. The Association for Computer Linguistics, 2009.

[14] Kevin Knight and Daniel Marcu. Summarization beyond sentence extraction: A probabilistic approach to sentence compression. *Artif. Intell.*, 139(1):91–107, 2002.

[15] Michael Mampaey, Jilles Vreeken, and Nikolaj Tatti. Summarizing data succinctly with the most informative itemsets. *TKDD*, 6(4):16:1–16:42, 2012.

[16] Kareem El Gebaly, Parag Agrawal, Lukasz Golab, Flip Korn, and Divesh Srivastava. Interpretable and informative explanations of outcomes. *PVLDB*, 8(1):61–72, 2014.

[17] Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Elisa Turricchia. Similarity measures for OLAP sessions. *Knowl. Inf. Syst.*, 39(2):463–489, 2014.

[18] Vitor Hirota Makiyama, Jordan Raddick, and Rafael D. C. Santos. Text mining applied to SQL queries: A case study for the SDSS skyserver. In Juan Antonio Lossio-Ventura and Hugo Alatrista-Salas, editors, *Proceedings of the 2nd Annual International Symposium on Information Management and Big Data - SIMBig 2015, Cusco, Peru, September 2-4, 2015.*, volume 1478 of *CEUR Workshop Proceedings*, pages 66–72. CEUR-WS.org, 2015.

[19] Kamel Aouiche, Pierre-Emmanuel Jouve, and Jérôme Darmont. Clustering-based materialized view selection in data warehouses. In Yannis Manolopoulos, Jaroslav Pokorný, and Timos K. Sellis, editors, *Advances in Databases and Information Systems, 10th East European Conference, ADBIS 2006, Thessaloniki, Greece, September 3-7, 2006, Proceedings*, volume 4152 of *Lecture Notes in Computer Science*, pages 81–95. Springer, 2006.

[20] Ting Xie, Oliver Kennedy, and Varun Chandola. Query log compression for workload analytics. *CoRR*, abs/1809.00405, 2018.

[21] Edwin T. Jaynes. Prior probabilities. *IEEE Trans. Systems Science and Cybernetics*, 4(3):227–241, 1968.

[22] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[23] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, March 2014.

[24] Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen P. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *J. Optimization Theory and Applications*, 169(3):1042–1068, 2016.

[25] Gokhan Kul, Shambhu J. Upadhyaya, and Varun Chandola. Detecting data leakage from databases on android apps with concept drift. In *IEEE TrustCom*, pages 905–913, 2018.

[26] Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[27] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, 2004.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[29] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Min. Knowl. Discov.*, 15(1):55–86, 2007.

[30] Oliver Kennedy, Jerry Antony Ajay, Geoffrey Challen, and Lukasz Ziarek. Pocket data: The need for TPC-MOBILE. In Raghunath Nambiar and Meikel Poess, editors, *Performance Evaluation and Benchmarking: Traditional to Big*

Data to Internet of Things - 7th TPC Technology Conference, TPCTC 2015, Kohala Coast, HI, USA, August 31 - September 4, 2015. Revised Selected Papers, volume 9508 of *Lecture Notes in Computer Science*, pages 8–25. Springer, 2015.

[31] G. Kul, D. T. A. Luong, T. Xie, V. Chandola, O. Kennedy, and S. Upadhyaya. Similarity metrics for sql query clustering. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2408–2420, Dec 2018.

[32] Sarika Mittal, Jothi Swarubini Vindhiya Varman, Gloria Chatzopoulou, Magdalini Eirinaki, and Neoklis Polyzotis. Querie: A query recommender system supporting interactive database exploration. In Wei Fan, Wynne Hsu, Geoffrey I. Webb, Bing Liu, Chengqi Zhang, Dimitrios Gunopulos, and Xindong Wu, editors, *ICDMW 2010, The 10th IEEE International Conference on Data Mining Workshops, Sydney, Australia, 13 December 2010*, pages 1411–1414. IEEE Computer Society, 2010.

[33] Arnaud Giacometti, Patrick Marcel, Elsa Negre, and Arnaud Soulet. Query recommendations for OLAP discovery-driven analysis. *IJDWM*, 7(2):1–25, 2011.

[34] Nodira Khoussainova, YongChul Kwon, Magdalena Balazinska, and Dan Suciu. Snipsuggest: Context-aware autocompletion for SQL. *PVLDB*, 4(1):22–33, 2010.

[35] Xiaoyan Yang, Cecilia M. Procopiuc, and Divesh Srivastava. Recommending join queries via query log analysis. In Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng, editors, *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 964–975. IEEE Computer Society, 2009.

[36] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. Stholes: A multidimensional workload-aware histogram. In Sharad Mehrotra and Timos K. Sellis, editors, *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 211–222. ACM, 2001.

[37] Ashish Kamra, Evimaria Terzi, and Elisa Bertino. Detecting anomalous access patterns in relational databases. *VLDB J.*, 17(5):1063–1077, 2008.

[38] Surajit Chaudhuri and Vivek R. Narasayya. An efficient cost-driven index selection tool for microsoft SQL server. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 146–155. Morgan Kaufmann, 1997.

[39] Sheldon J. Finkelstein, Mario Schkolnick, and Paolo Tiberio. Physical database design for relational databases. *ACM Trans. Database Syst.*, 13(1):91–128, 1988.

[40] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 496–505. Morgan Kaufmann, 2000.

[41] Ling Hu, Kenneth A. Ross, Yuan-Chi Chang, Christian A. Lang, and Donghui Zhang. Queryscope: visualizing queries for repeatable database tuning. *PVLDB*, 1(2):1488–1491, 2008.

[42] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, 1994.

[43] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2):6, 2006.

[44] Daniel J. Abadi, Samuel Madden, and Miguel Ferreira. Integrating compression and execution in column-oriented database systems. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 671–682. ACM, 2006.

[45] Gennady Antoshenkov and Mohamed Ziauddin. Query processing and optimization in oracle rdb. *VLDB J.*, 5(4):229–237, 1996.

[46] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. Compressing bitmap indexes for faster search operations. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management, July 24-26, 2002, Edinburgh, Scotland, UK*, pages 99–108. IEEE Computer Society, 2002.

[47] Alistair Moffat and Justin Zobel. Compression and fast indexing for multi-gigabyte text databases. *Australian Computer Journal*, 26(1):1–9, 1994.

[48] Sihem Amer-Yahia and Theodore Johnson. Optimizing queries on compressed bitmaps. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 329–338. Morgan Kaufmann, 2000.

[49] G. Antoshenkov. Byte-aligned bitmap compression. In *Proceedings of the Conference on Data Compression*, DCC '95, pages 476–, Washington, DC, USA, 1995. IEEE Computer Society.

[50] Przemyslaw Skibinski and Jakub Swacha. Fast and efficient log file compression. In Yannis E. Ioannidis, Boris Novikov, and Boris Rachev, editors, *Communications of the Eleventh East-European Conference on Advances in Databases and Information Systems, Varna, Bulgaria, September 29 - October 3, 2007*, volume 325 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[51] Nicolas Bruno and Surajit Chaudhuri. Automatic physical database tuning: A relaxation-based approach. In *ACM SIGMOD*, 2005.

[52] Oliver Kennedy, Jerry Ajay, Geoffrey Challen, and Lukasz Ziarek. Pocket Data: The need for TPC-MOBILE. In *TPC-TC*, 2015.

[53] Gokhan Kul, Duc Luong, Ting Xie, Patrick Coonan, Varun Chandola, Oliver Kennedy, and Shambhu Upadhyaya. Ettu: Analyzing query intents in corporate databases. In *WWW Companion*, 2016.

[54] Cynthia Dwork. Differential privacy. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*. Springer, 2006.

[55] Carsten Sapia. Promise: Predicting query behavior to enable predictive caching strategies for OLAP systems. In *DaWaK*, 2000.

[56] Arnaud Giacometti, Patrick Marcel, Elsa Negre, and Arnaud Soulet. Query recommendations for OLAP discovery driven analysis. In *ACM DOLAP*, 2009.

[57] X. Yang, C. M. Procopiuc, and D. Srivastava. Recommending join queries via query log analysis. In *IEEE ICDE*, 2009.

[58] Kostas Stefanidis, Marina Drosou, and Evaggelia Pitoura. "You May Also Like" results in relational databases. In *ACM DOLAP*, 2009.

[59] Nodira Khoussainova, YongChul Kwon, Magdalena Balazinska, and Dan Suciu. SnipSuggest: context-aware autocompletion for SQL. *pVLDB*, 2010.

[60] Gloria Chatzopoulou, Magdalini Eirinaki, Suju Koshy, Sarika Mittal, Neoklis Polyzotis, and Jothi Swarubini Vindhiya Varman. The QueRIE system for personalized query recommendations. *IEEE Data Eng. Bull.*, 2011.

[61] Wolfgang Gatterbauer. Databases will visualize queries too. *pVLDB*, 2011.

[62] Julien Aligon, Kamal Boulil, Patrick Marcel, and Verónika Peralta. A holistic approach to OLAP sessions composition: The falseto experience. In *ACM DOLAP*, 2014.

[63] Kamel Aouiche, Pierre-Emmanuel Jouve, and Jérôme Darmont. Clustering-based materialized view selection in data warehouses. In *ADBIS*, 2006.

[64] Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Elisa Turricchia. Similarity measures for OLAP sessions. *Knowledge and information systems*, 2014.

[65] Vitor Hirota Makiyama, M Jordan Raddick, and Rafael DC Santos. Text mining applied to SQL queries: A case study for the SDSS SkyServer. In *SIMBig*, 2015.

[66] Mohammed J Zaki and Wagner Meira Jr. *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, 2014.

[67] Ashish Kamra, Evimaria Terzi, and Elisa Bertino. Detecting anomalous access patterns in relational databases. *VLDBJ*, 2007.

[68] Sunu Mathew, Michalis Petropoulos, Hung Q. Ngo, and Shambhu Upadhyaya. A data-centric approach to insider attack detection in database systems. In *RAID*, 2010.

[69] Hoang Vu Nguyen, Klemens Böhm, Florian Becker, Bertrand Goldman, Georg Hinkel, and Emmanuel Müller. Identifying user interests within the data space-a case study with skyserver. In *EDBT*, 2015.

[70] Rakesh Agrawal, Ralf Rantzau, and Evimaria Terzi. Context-sensitive ranking. In *ACM SIGMOD*, 2006.

[71] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, 1977.

[72] Bikash Chandra, Mathew Joseph, Bharath Radhakrishnan, Shreevidhya Acharya, and S. Sudarshan. Partial marking for automated grading of SQL queries. *VLDBj*, 9(13):1541–1544, September 2016.

[73] Carsten Sapia. On modeling and predicting query behavior in OLAP systems. In *DMDW*, 1999.

[74] P. Seshadri, H. Pirahesh, and T. Y. C. Leung. Complex query decorrelation. In *IEEE ICDE*, 1996.

[75] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *pVLDB*, 1995.

[76] Fei Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *pVLDB*, 2014.

[77] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *JMLR*, 2011.

[78] Yuqing Sun, Haoran Xu, Elisa Bertino, and Chao Sun. A data-driven evaluation for insider threats. *Data Science and Engineering*, 1(2):73–85, 2016.

[79] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *DMKD*, 2004.