Statistical Schema Learning using Occam's Razor

Justin Talbot*
justin.talbot@databricks.com
Databricks
San Francisco, CA, USA

Daniel Ting*
dting.tr@gmail.com
Tableau Research
Seattle, WA, USA

ABSTRACT

A judiciously normalized database schema can increase data interpretability, reduce data size, and improve data integrity. However, real world data sets are often stored or shared in a denormalized state. We examine the problem of automatically creating a good schema for a denormalized table, approaching it as an unsupervised machine learning problem which must learn an optimal schema from the data. This differs from past rule-based approaches that focus on normalization into a canonical form. We define a principled schema optimization criterion, based on Occam's razor, that is robust to noise and extensible—allowing users to easily specify desirable properties of the resulting schema. We develop an efficient learning algorithm for this criterion and empirically demonstrate that it is 3 to 100 times faster than previous work and produces higher quality schemas with $1/5^{th}$ the errors.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Information systems** → **Database design and models**.

KEYWORDS

database normalization, information theory, Bayesian learning

ACM Reference Format:
Justin Talbot and Daniel Ting. 2022. Statistical Schema Learning using Occam's Razor. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3514221.3526174

1 INTRODUCTION

The benefits of schema normalization are well-known: it decreases data redundancy, reducing storage requirements; it makes data integrity constraints easier to enforce; and it can improve the interpretability and usability of the data when tables in the schema correspond to meaningful domain concepts. However, often a single, denormalized table is easier to share and query. Thus, a common challenge when beginning an analysis or data modeling project is to decompose a denormalized data set into a good normalized schema. This can be difficult and time consuming to do manually.

To address this, we tackle automatically recovering a lossless, normalized, snowflake-shaped schema from a denormalized input table. We propose a probabilistic, machine learning model and an accompanying unsupervised learning algorithm with advantages in robustness, quality, and speed over existing methods. Specifically,

(1) It chooses a quantitatively best schema from a broad search space, naturally selecting schemas that are more likely to contain interpretable entities and relationships.
(2) It is robust to randomness in the data, handling both false negatives (functional dependencies obscured by noise in the data) and false positives (functional dependencies in the data that do not correspond to real-world dependencies).
(3) It provides a principled method to prefer schema with desirable properties, like the existence of simple candidate keys.
(4) It can be orders of magnitude faster than existing methods.

Our approach achieves this by directly optimizing a principled measure of schema quality. This differs from existing schema normalization methods which focus on satisfying technical conditions imposed by canonical normal forms, but which do not necessarily lead to concise, well-organized schemas. Further, satisfying these conditions requires discovering functional dependencies in the data, an expensive task that our approach avoids.

In this paper, we make several methodological and theoretical contributions. We show how to apply Occam's razor, the principle that the simplest explanation is often the best, to the schema learning problem, leading to a principled schema quality metric. We then embed this approach in a Bayesian probabilistic framework that makes it easy to optimize for additional desirable schema properties, such as having concise primary keys. Coupled with these contributions, we develop an efficient algorithm for finding the optimal schema. We run experiments on real-world datasets demonstrating substantial improvements in robustness, quality, and speed.

2 RELATED WORK AND CHALLENGES

Standard methods for manual schema normalization [4, 6, 10] rely on a two-stage process of (1) specifying functional dependencies (FDs) that hold in a relation and (2) using these FDs to decompose the relation to a desired normal form. Papenbrock and Naumann [30] leverage recent progress in FD discovery algorithms to automate this process, producing lossless snowflake schemas in Boyce-Codd Normal Form (BCNF). Kenig et al. [20] relax the snowflake requirement by finding approximate Multivalued Dependencies, then using them to produce lossy acyclic schemas.

We find two practical challenges with these two-stage approaches. First, a small number of errors in a data set can obscure functional dependencies making it impossible to recover important entities, and, conversely, accidental dependencies often occur by chance, leading to normalized schemas with tables that do not correspond to real world entities (Figure 1). This latter issue is unsurprising as the

*This work was done while the authors were employed at Tableau Software.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3526174

ID	Time	UserID	Name
1	T_1	U_1	Joe
2	T_2	U_1	Joe
3	T_2	U_3	Joe
4	T_3	U_2	Sue
5	T_4	U_2	Sue
6	T_5	U_1	Joe
7	T_6	U_3	Jo

(a) Denormalized table, $\mathcal{D}$ (error on line 7)

ID	Time	FK
1	T_1	r_1
2	T_2	r_1
3	T_2	r_3
4	T_3	r_2
5	T_4	r_2
6	T_5	r_1
7	T_6	r_4

$\bowtie$

UserID	Name
U_1	Joe
U_2	Sue
U_3	Joe
U_3	Jo

(b) Desired normalization, despite error

ID	UserID	FK
1	U_1	r_1
2	U_1	r_2
3	U_3	r_2
4	U_2	r_3
5	U_2	r_4
6	U_1	r_5
7	U_3	r_6

$\bowtie$

Time	Name
T_1	Joe
T_2	Joe
T_3	Sue
T_4	Sue
T_5	Joe
T_6	Jo

(c) Semantically meaningless normalization

Figure 1: Example of difficulties in schema learning from data dependencies. Given $\mathcal{D}$, we would like to learn the schema in (b) (the foreign key column references the row number in the second table), but an error in row 7 ("Jo" instead of "Joe") obscures the functional dependency between UserID and Name. Further, Time, due to its high cardinality, inadvertently functionally determines Name (but not UserID), which can result in two-stage normalization approaches producing the schema in (c), incorrectly separating UserID and Name.

number of potential FDs, and hence the number of accidental dependencies, grows exponentially with the number of columns. While previous work develops methods for finding approximate dependencies [16, 21, 36] and ignoring spurious dependencies [7, 23, 24], how these methods and their hyperparameters can be calibrated so that the resulting FDs are meaningful for automated schema normalization remains unexplored. Current schema normalization methods [30] try to mitigate the problem of accidental dependencies by applying heuristics to prioritize FDs; however, this results in lower quality schemas than those found by our method. Second, enumerating data-derived dependencies remains computationally costly despite continued algorithmic advances [1, 12, 16, 22, 26, 29, 34, 35]. In two-stage approaches, most of this computation is wasted as the vast majority of functional dependencies found in the search stage will not be used later in normalization.

We instead use an information theoretic approach to identify an optimal schema. Previous work [2, 20, 21, 24, 30, 36] has used information theoretic approaches to measure and exploit dependence between columns. However, these works make the traditional assumption that all rows in the data are independent draws from a single, fixed underlying distribution. In our work, each table in a normalized schema corresponds to a different random distribution. Rows in the denormalized table are drawn from these shared distributions. These rows are correlated since changing one of the underlying distributions changes multiple rows. Our model can be seen as exploiting the dependence of columns *and rows* to determine a better schema decomposition. From a statistical perspective, our approach learns a graphical model with a tree structure, similar to the Chow-Liu algorithm [9]. However, it differs from Chow-Liu and past literature on structure learning [11, 18] which also assume rows are independent and identically distributed.

Our work has a side effect of producing a compressed representation of the input data, so is also related to table compression methods. Previous work models the joint distribution over columns [14, 31] and captures fine-grained dependencies [15, 17] to produce a more efficient encoding, while our method captures dependencies that are useful for schema normalization.

3 SCHEMA LEARNING OVERVIEW

In contrast to FD-driven approaches to schema learning, we formulate schema learning as an unsupervised machine learning problem that directly searches for an *optimal* schema. This requires three components: (1) an optimization search space that includes plausible schemas, (2) a principled objective function that assesses the quality of a schema, and (3) an algorithm that can solve the optimization problem efficiently.

Section 4 describes our optimization search space—*conceptual snowflake schemas*, in which tables are defined, but primary keys are not identified. This space is shown to be equivalent to a multilevel clustering of the columns. To evaluate schema quality, Section 5 shows how Occam's razor can be used to derive a principled objective function based on the minimum description length (MDL) that naturally rewards schemas that more concisely capture the entities within the data. We then develop this into a Bayesian model that provides a formulaic means for users to incorporate other desirable schema properties into the objective.

However, the resulting optimization problem is non-trivial. The optimization is over a discrete space that grows super-exponentially in size with the number of columns, and the objective function is expensive to evaluate. Section 6 provides an efficient algorithm that combines branch-and-bound, dynamic programming, and a greedy heuristic. This yields exact optimization of moderately sized problems and approximate solutions for large problems. Section 7 shows we obtain both higher quality results than the state-of-the-art and runtimes up to 2 orders of magnitude faster. To avoid technical details in the exposition, we defer all proofs to the appendix.

4 OPTIMIZATION SEARCH SPACE

Given a denormalized table, our goal is to search for a normalized schema where its tables represent interpretable real-world entities. The search space must be broad enough to capture such schemas, but also restricted enough to make optimization tractable. Acyclic schemas have many good properties for data modeling [5]. However, the space of such schemas is large and, in practice, they seldom admit lossless join decompositions in the presence of noise or missing rows. Instead, we focus on the more restricted space of snowflake schemas which is also used in previous work [30]. By snowflake schema, we mean a schema with an entity-relation diagram that forms an n-ary rooted tree with many-to-one relationships from parent tables to children. In contrast to general acyclic schemas, the restricted space of snowflake schemas always admit trivial lossless join decompositions even for noisy data. Further, snowflake

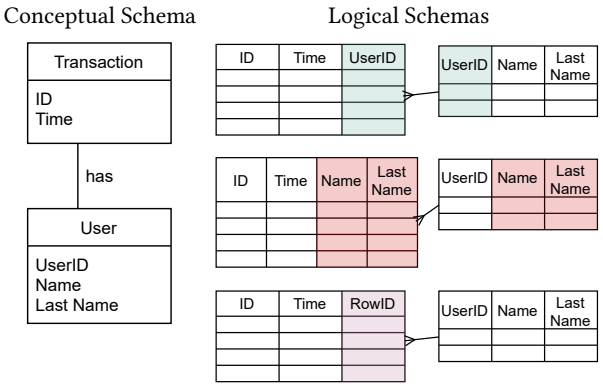


Figure 2: Conceptual and logical schemas. A single conceptual snowflake schema (left) can be represented as multiple logical schemas (top right) based on the choice of primary key (colored). To avoid the cost of selecting a primary key, our approach only searches over schemas which use the physical row index as the key (bottom right). This row primary key column is implicit and does not need to be materialized.

schemas can often use junction tables to model many-to-many and one-to-many relationships in acyclic schemas, reducing the gap in representational power. Finally, we observe that denormalizing many-to-many relationships results in an explosion in data size; thus, we believe that denormalized tables from acyclic schemas are less likely to arise in practice than from snowflake schemas.

In FD-based schema normalization, primary keys for the resulting schema are identified as part of the normalization process. The left side of a FD used to split out a table defines a primary key for that table. In contrast, we do not try to identify a primary key from the set of candidate keys during the search. To generate a lossless join decomposition without identifying primary keys, we simply use row numbers to link tables (Figure 2). This approach is motivated by two considerations. First, a concise primary key for an entity may not exist in the denormalized table or it may be corrupted by noise. Using row numbers as a stand-in for the primary key allows our algorithm to robustly handle these cases. Second, not identifying primary keys simplifies the search space. Since we do not need to duplicate input columns to create FK-PK pairs, each column in the input table appears exactly once in the output schema. This means that a snowflake schema can be seen as a hierarchical partitioning of the input columns. In Section 6, we leverage this property to create an efficient algorithm for finding an optimal schema. (Note that a primary key detection algorithm [19] could be run afterwards to add primary keys to our schemas.)

While we have constrained our search space to make the optimization problem more tractable, the space is still large. Since our space of snowflake schemas is equivalent to that of hierarchical partitionings of the input columns, the search space is super-exponential in the number of columns, m.

Theorem 1. *Let Γ be the set of snowflake schemas in the search space of our optimization function. The space of schemas grows super-exponentially, and* $\log|\Gamma| = \Omega(m \log m)$.

This is substantially larger than the space of possible functional dependencies. So, compared to previous work, we've nominally increased the difficulty of the problem. However, this optimization framing permits more efficient algorithms and better normalization.

5 OPTIMIZATION OBJECTIVE

We next present our optimization objective using two perspectives—an information theoretic one and a statistical one. The information theoretic perspective is simple and tied to the familiar database concept of physical design. Inspired by Occam's razor, we posit that schemas that lead to simpler descriptions of the data are generally better. To quantify simplicity, we use the number of bits needed to store the data. Since a schema influences the physical layout of the data, and hence its size in bits, each schema can be assigned a measure of simplicity—the size of the data when losslessly decomposed according to the schema—and an optimal schema can be chosen that minimizes that measure.

Rather than measuring the data size using a particular encoding scheme or compression algorithm, we use the information theoretic *entropy* since it is a lower bound on the data size. This is incorporated into a minimum description length measure that includes the both the size of the decomposed data as well as the size of the compression codes and any necessary metadata.

This information theoretic approach converts the problem of schema learning into an optimization problem, but it is heuristic in nature, and focuses on only one benefit of schema design—reduced storage. So we also present a second perspective using probabilistic generative models that formalizes and generalizes the method, allowing us to extend our objective to capture other benefits of good schema design, such as the existence of simple primary keys.

In particular, our information theoretic approach is a specific instance of a statistical modeling and model selection problem. Since the definition of entropy is $\mathbb{E}p(X)\log_2 p(X)$, the information theoretic approach requires making assumptions about and estimating an underlying probability distribution. By first identifying the underlying generative model for each schema, we can extend it to a Bayesian model where each schema is assigned a probability representing the belief that a given schema is the correct one.

5.1 Information Theoretic Interpretation

Each possible snowflake schema, $\mathcal{S}$, in our search space corresponds to a lossless join decomposition of the input data, $\mathcal{D}$. Our goal is to select the schema which best captures the structure—the entities and relationships—within the data. Applying the intuition of Occam's razor suggests that schemas that correspond to simpler and more compact join decompositions better capture the structure of the input data. We can formalize this intuition as a minimum description length (MDL) objective, J, of the form:

$$J_{MDL}(\mathcal{S}|\mathcal{D}) := Length(\mathcal{D}|\mathcal{S}) + Length(\mathcal{S}) \quad (1)$$

where $Length(\mathcal{D}|\mathcal{S})$ is the length of the data when stored in the join decomposition determined by $\mathcal{S}$, and $Length(\mathcal{S})$ is the length of the schema. This objective favors join decompositions that concisely represent the data, while penalizing overly complex schemas.

To make this objective concrete, $Length(\mathcal{D}|\mathcal{S})$ is total size of the tables in the schema's join decomposition. We assume that both data and foreign key columns are stored with an ideal entropy

encoding; thus, the size of each column, c, is its empirical entropy, $H(c)$, times the number of rows in its corresponding table, n_t:

$$Length(\mathcal{D}|\mathcal{S}) := \sum_{t \in Tbls(\mathcal{S})} \sum_{c \in Cols(t)} n_t \cdot H(c) \quad (2)$$

The length of a schema, $Length(\mathcal{S})$, is the total cost of storing the compression codes for each column in the join decomposition (including any foreign key columns):

$$Length(\mathcal{S}) := \sum_{t \in Tbls(\mathcal{S})} \sum_{c \in Cols(t)} \beta \cdot |\Omega_c| = \beta \left(\sum_{t \in \mathcal{S}} n_t \right) + const \quad (3)$$

where Ω_c is the set of unique values in column c and β represents the cost of storing one code entry. The simplification into the total number of rows in the schema follows from the fact that only the domains of FKs can differ between schemas.

While this metric is based on the compressed length of the input data, our goal is not only compression. The key MDL-based insight is that a schema which compresses the data well must necessarily also concisely capture the important structure in the data.

5.2 Probabilistic Interpretation Overview

Under the information theoretic interpretation in the previous section, our approach finds the schema that minimizes data size. In this section, we give an equivalent probabilistic interpretation, under which our approach finds the schema which has the maximum regularized likelihood of generating the input denormalized table.

This framing provides a better understanding of the assumptions made by the MDL objective, a way to cast the model selection problem as a pure optimization problem that can be solved efficiently, and a framework for incorporating prior information about the true schema. We note that optimization-based model selection with complexity penalties are not new and can sometimes be viewed as approximations to fully Bayesian methods [33].

Our goal is to produce a good posterior distribution $p(\mathcal{S}|\mathcal{D})$ over possible schema given the data. Similar to the information theoretic MDL objective, this posterior can be maximized to find an optimal, maximum a posteriori (MAP) schema. Building this posterior requires two components, a generative model that defines the likelihood of the data given a schema, $p(\mathcal{D}|\mathcal{S})$, and a prior, $\pi(\mathcal{S})$, that prefers schema with good properties. The resulting posterior distribution is $p(\mathcal{S}|\mathcal{D}) \propto p(\mathcal{D}|\mathcal{S})\pi(\mathcal{S})$ by Bayes' rule.

To create a generative model, we can treat a snowflake schema as a hierarchical graphical model which encodes the conditional dependence between sets of columns (Figure 3). This leads to a straightforward generation procedure where parent tables are generated conditional on their children. However, to form a full generative hierarchical model, an additional set of parameters, ρ, are needed to specify how to generate the data within each table. Since only the schema is of interest in schema learning, these ρ are called nuisance parameters.

Thus, in our probabilistic interpretation we must (1) specify the full generative distribution $p(\mathcal{D}|\mathcal{S}, \rho)$, (2) deal with the nuisance parameters ρ, and (3) specify a prior distribution, $\pi(\mathcal{S})$. In Section 5.3, we show that the MDL objective given in Equation 1 is equivalent to a specific solution for each of these issues. Then in Section 5.4, we leverage our probabilistic framework to extend our objective to include useful prior information.

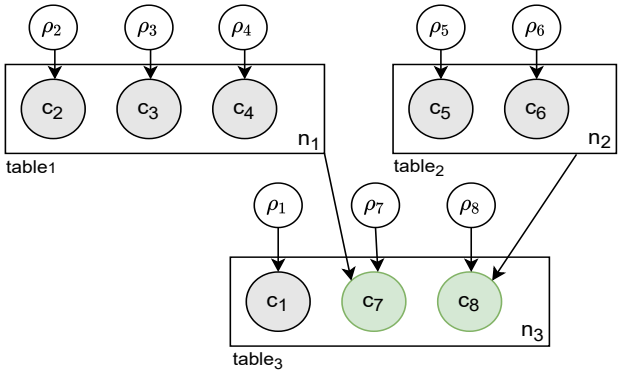


Figure 3: Graphical model for a schema with 3 tables, 6 data columns (c_1–c_6) and 2 foreign key columns (c_7 and c_8). Nuisance parameters ρ define the distribution of values in each column. A denormalized table can be generated from this model by first generating $table_1$ and $table_2$, then independently drawing n_3 entries for column $X^1_{:,1}$ from distribution ρ_1 and n_3 tuples from c_1 and $table_2$ with probabilities ρ_7 and ρ_8 respectively. Here, column c_j in table t is the same as $X^t_{:j}$ in the normalized dataset.

5.3 Probabilistic Interpretation Details

To generate a denormalized table, $\tilde{X}$, from a schema, $\mathcal{S}$, we treat the schema as a hierarchical graphical model, as illustrated in Figure 3. (We use $\tilde{X}$ here to distinguish a random denormalized table drawn from our model from the observed data $\mathcal{D}$). Our generation procedure works by drawing a table instance X^t for each table t in the schema $\mathcal{S}$; these tables are then joined create $\tilde{X}$. The number of rows n_t for each X^t is considered fixed in the model as is the domain Ω_c of each column c. The unknown parameters are probability distributions ρ_c for each column c, which can be either data columns or foreign key columns. We draw a table, X^t, with the following data generating process:

- For each data column c in t, independently draw n_t independent values from the distribution ρ_c. In other words, $X^t_{ic} \overset{iid}{\sim} \rho_c$ for $i \leq n_t$.
- For each foreign key column c in t, similarly draw $X^t_{ic} \overset{iid}{\sim} \rho_c$ for $i \leq n_t$.

The denormalized table, $\tilde{X}$, is formed by joining the all the X^t. The likelihood of this table can be written as:

$$p(\tilde{X}|\mathcal{S}, \rho) = \prod_{t \in Tbls(\mathcal{S})} \prod_{c \in Cols(t)} \prod_{i=1}^{n_t} \rho_c(X^t_{ic}). \quad (4)$$

While the representation using FKs leads to all columns in X^t being independent, the graphical model in Figure 3 represents the process for generating the denormalized table $\tilde{X}$. If tables 1, 2, and 3 respectively represent customer, stores, and transactions, then each transaction row's information can only be filled out once its customer's and store's information are known.

Maximizing the log of this likelihood is equivalent to minimizing a portion of the MDL objective.

Theorem 2. *Under the above generative model, a schema, $\mathcal{S}$, has a maximum log-likelihood of*

$$-J_{MLE}(\mathcal{S}|\mathcal{D}) := \sup_{\rho} \ell(\mathcal{S}, \rho; \mathcal{D}) = -Length(\mathcal{D}|\mathcal{S}) \tag{5}$$

where $\rho = \{p_c(\cdot|\mathcal{S})\}_{c \in Cols(\mathcal{S})}$ encodes every column's empirical distribution.

Here, the log-likelihood is defined by $\ell(\mathcal{S}, \rho; \mathcal{D}) := \log p(\mathcal{D}|\mathcal{S}, \rho)$ and turns the conditional probability, which is a function of the data, into a function of the parameters. Maximizing over the nuisance parameters ρ turns it into a objective function on the schema only. For each column c, the ρ_c that maximizes this log-likelihood is the observed, empirical distribution of the column after normalizing the data into schema $\mathcal{S}$. For example, in Figure 1b's schema, the distribution over Name assigns probability 1/4 to 'Jo', not the probability 1/7 given in the denormalized table in Figure 1a.

This generative formulation lets us cast schema learning as Bayesian estimation problem and allows us to inject prior beliefs about the correct or best schema. If the prior on the schema, $\mathcal{S}$, does not depend on the nuisance parameters, $\pi_0(\mathcal{S}, \rho) \propto \pi(\mathcal{S})$, then we obtain a more general form of the MDL objective.

$$J(\mathcal{S}|\mathcal{D}) := -\max_{\rho} \log_2 p(\mathcal{S}, \rho|\mathcal{D}) = J_{MLE}(\mathcal{S}|\mathcal{D}) - \log_2 \pi(\mathcal{S}) \tag{6}$$

The MDL objective from the previous section, J_{MDL}, is recovered when the prior depends only on the length of the schema: $\log_2 \pi(\mathcal{S}) = -Length(\mathcal{S}) + const$.

5.4 Constructing useful priors

Priors allow us to tune our objective to favor schemas with useful properties beyond having a short description length. However, the form in Equation 6 is too general, providing no guidance on how to encode useful properties or how to scale the prior and allowing for priors that can make optimization difficult. In this section, we provide templates for generating priors that allow for efficient optimization of the objective and have a meaningful interpretation that aids practitioners when setting the parameters.

Given a schema, $\mathcal{S}$, and table, $t \in Tbls(\mathcal{S})$, let $\tilde{D}_t$ denote the table reconstructed by joining t and its descendants. To enable efficient optimization of the objective, we consider log priors of the form:

$$\log_2 \pi(\mathcal{S}) = \sum_{t \in Tbls(\mathcal{S})} g(parent(t), \tilde{\mathcal{D}}_t). \tag{7}$$

Our learning algorithm performs a hierarchical partitioning of the columns, and priors of this form utilize only the information available immediately before and after a partitioning step. We note that priors of this form include those of the form $\log_2 \pi(\mathcal{S}) = \sum_{t \in Tbls(\mathcal{S})} g(t)$ which looks only at individual tables and not relationships in the schema.

Our basic setup treats the negative log prior, $-\log_2 \pi$, similarly to a soft constraint in an optimization problem. A penalized objective that discourages a table t from having property $\mathcal{P}$ is given by $J_{MLE}(\mathcal{S}|\mathcal{D}) + \lambda \mathbb{I}(t \in \mathcal{P})$ where $\lambda \geq 0$ and $\mathbb{I}(t \in \mathcal{P})$ is the indicator function that is 1 when t satisfies P and 0 otherwise. Since the objective is increased anytime t has the unwanted property P, such schemas are at a disadvantage at being picked.

We consider 3 useful log priors of this form that serve different purposes. These (1) penalize tables with undesirable properties, (2) discourage splits, and (3) encourage desirable splits:

$$\log_2 \pi(\mathcal{S}) = -\sum_{t \in Tbls(\mathcal{S})} \gamma \cdot \mathbb{I}(t \in \mathcal{P}) \cdot n_t \qquad \text{(Table penalty)}$$

$$\log_2 \pi(\mathcal{S}) = -\sum_{t \in Tbls(\mathcal{S})} \gamma \cdot \mathbb{I}(\tilde{D}_t \in \mathcal{P}) \cdot n_{pa(t)} \qquad \text{(Discourage)}$$

$$\log_2 \pi(\mathcal{S}) = \sum_{t \in Tbls(\mathcal{S})} \tau \cdot \mathbb{I}(t \in \mathcal{P}) \cdot n_t \log_2 n_t \qquad \text{(Encourage)}$$

where $\gamma > 0$ and $\tau \in [0, 1)$ are user specified weights. Note that for the split discouragement prior, the indicator can only check properties about $\tilde{D}_t$, which contains no information about what columns are further split out, rather than properties about t.

The row count terms scale the parameters γ and τ to have intuitive interpretations under the MDL and Bayesian frameworks. The penalties with γ are equivalent to adding a column with entropy γ to either the table or its parent. The encouragement scaling τ eliminates the fraction τ from the overhead of creating a new table. The behavior of rewarding a property by reducing the overhead distinguishes it from penalization methods and makes it not expressible as a soft constraint. Although the table penalty can mimic the encouragement behavior by applying a penalty when some property is *not* satisfied, the penalty has a side effect of strengthening the MDL prior, which may not be desirable. This is since the default behavior of penalizing by the number of rows is precisely what the MDL penalty does in Equation 3.

We note that other prior forms can also support efficient optimization. For example, in some cases, it may be sensible to apply a constant penalty when a property is satisfied or use some other scaling that grows sublinearly with the table size. Similarly, the indicator function could be replaced with a smooth function.

Given a collection of priors, $\log_2 \pi_i$, in one of the forms in Equation Table penalty, we can combine them to form our final prior,

$$\log_2 \pi(\mathcal{S}) = \sum_{\mathcal{P}} \log_2 \pi_i(\mathcal{S}) + const. \tag{8}$$

We define a *canonical prior* to be one of this form with an additional constraint that the sum over the scaling parameter τ_i for encouragement priors satisfies $\tau_{tot} = \sum_i \tau_i \in [0, 1)$. Later, in Theorem 4, we show that such priors lead to an efficient optimization algorithm. The following examples demonstrate properties that can be captured with the above priors:

Example 1: Columns with names sharing a common prefix likely come from the same source entity. This can be encoded as a table penalty when the columns do not appear together. If the columns sharing a common prefix are $\mathcal{F}$, then the property can be checked by setting $\mathbb{I}(t \in \mathcal{P}) = \mathbb{I}(\,|\mathcal{F} \cap Cols(t)| \notin \{0, |\mathcal{F}|\}\,)$.

Example 2: Columns commonly queried together are likely from the same entity. This scenario is similar to Example 1 but uses a different set of columns for $\mathcal{F}$. In this case, it may be sensible to set the scaling parameter, γ, proportional to the number of times the columns are queried together. This allows workload or performance information to influence the choice of schema.

Example 3: Tables are likely to have a simple candidate key. This can be encoded using a combination of encouragement and discouragement priors. A discouragement prior penalizes creating a table with no simple candidate keys. An encouragement prior rewards

creating a table with a simple candidate key, even if it has a close to 1-to-1 relationship with the parent and does not substantially reduce the redundancy in the data.

6 LEARNING ALGORITHM

Algorithm 1: SSLearn

Input: Denormalized table, $\mathcal{D}$
Output: Optimal snowflake schema, $\mathcal{S}$

Memoized Function `Optimize`($\mathcal{D}$):
  // Initialize queue and upper bound with empty schema
  let q = new PriorityQueue(orderby = $J(\mathcal{S})$)
  q.push({$\mathcal{S}$: $\emptyset$, C_{parent}: [], C_{child}: [], C_{tail}: $Cols(\mathcal{D})$})
  let upper = `UpperBound`($\emptyset$, $\mathcal{D}$)
  while q is not empty **do**
    let {$\mathcal{S}$, C_{parent}, C_{child}, C_{tail}} = q.pop()
    // If all columns have been partitioned, return
    if C_{tail} = [] **then**
      return $\mathcal{S}$
    // Apply bounds to prune search space
    if `LowerBound`($\mathcal{S}$, $\Pi^{+}_{C_{tail}}\mathcal{D}$) > upper **then**
      continue
    // Otherwise, branch on next column in table
    [c_{head} | C'_{tail}] = C_{tail}
    // Branch with c_{head} partitioned to parent table
    C'_{parent} = `append`(C_{parent}, c_{head})
    $\mathcal{S}_0$ = `Normalize`($\mathcal{D}$, C'_{parent}, C_{child})
    q.push({$\mathcal{S}_0$, C'_{parent}, C_{child}, C'_{tail}})
    // Branch with c_{head} partitioned to child table
    C'_{child} = `append`(C_{child}, c_{head})
    $\mathcal{S}_1$ = `Normalize`($\mathcal{D}$, C_{parent}, C'_{child})
    q.push({$\mathcal{S}_1$, C_{parent}, C'_{child}, C'_{tail}})
    // Use prefix solutions to improve upper bound
    upper = min(upper, `UpperBound`($\mathcal{S}_0$, $\Pi^{+}_{C'_{tail}}\mathcal{D}$)),
     `UpperBound`($\mathcal{S}_1$, $\Pi^{+}_{C'_{tail}}\mathcal{D}$))

Function `Normalize`($\mathcal{D}$, C_{parent}, C_{child}):
  let $\mathcal{D}_{parent}$ = $\Pi^{+}_{C_{parent}}(\mathcal{D})$, $\mathcal{D}_{child}$ = $\Pi_{C_{child}}(\mathcal{D})$
  if $\mathcal{D}_{parent}$ = $\mathcal{D}$ or $\mathcal{D}_{child}$ = $\mathcal{D}$ **then**
    return $\mathcal{D}$
  else
    $\mathcal{S}_{parent}$ = `Optimize`($\mathcal{D}_{parent}$)
    $\mathcal{S}_{child}$ = `Optimize`($\mathcal{D}_{child}$)
    return `LinkWithFK`($\mathcal{S}_{parent}$, $\mathcal{S}_{child}$)

Optimizing our objective requires a combinatorial search over a super-exponentially large search space. To make this tractable, first, we propose a global learning algorithm, SSLearn, that structures the search as a hierarchical partitioning of the columns which allows us to combine dynamic programming and branch-and-bound methods to perform an efficient search that usually does not need to enumerate all possible schemas. Second, we extend SSLearn to an anytime algorithm that can return good, if not optimal, solutions at any point in time.

6.1 SSLearn

The SSLearn algorithm enumerates the space of snowflake schemas over an input table $\mathcal{D}$ by recursively proposing binary partitions of the table's columns (see Figure 4). Each partition creates a new child table in the proposed snowflake. The child table is deduplicated while the parent table is not, so a many-to-one join between the two tables losslessly recovers the input table. Recursive partitioning of both the parent table and child table can generate arbitrary snowflake schemas. We can find the optimal snowflake by finding the optimal column partitioning at each step. Formally:

Theorem 3. *Let $J(\mathcal{S}|\mathcal{D})$ be an objective function in the form in Equation 6 with a prior of the form in Equation 7. Partition $Cols(\mathcal{D})$ into two disjoint sets, C_{parent} and C_{child}, and denote the duplicate-preserving projection—$\mathcal{D}_{parent} = \Pi^{+}_{C_{parent}}\mathcal{D}$, and the deduplicating projection—$\mathcal{D}_{child} = \Pi_{C_{child}}\mathcal{D}$. Then,*

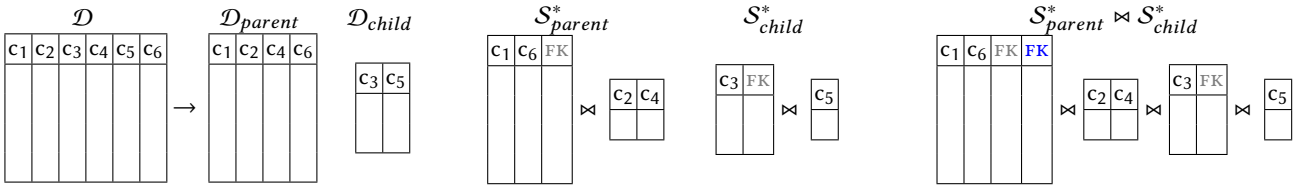
$$\min_{\mathcal{S}} J(\mathcal{S}|\mathcal{D}) = \min_{C_{parent}, C_{child}} \Big(w(\mathcal{D}_{parent}, \mathcal{D}_{child}) + \min_{\mathcal{S}_{parent}} J\left(\mathcal{S}_{parent} \mid \mathcal{D}_{parent}\right) + \min_{\mathcal{S}_{child}} J\left(\mathcal{S}_{child} \mid \mathcal{D}_{child}\right) \Big) \tag{9}$$

where w is a function that does not depend on $\mathcal{S}$.

An implementation of this minimization is sketched in Algorithm 1. `Optimize` uses a branch-and-bound search to find an optimal partitioning of $Cols(\mathcal{D})$ into C_{parent} and C_{child} (the outer minimization of Equation 3). Given a partitioning, `Normalize` splits $\mathcal{D}$ into the two subtables, $\mathcal{D}_{parent}$ and $\mathcal{D}_{child}$. If the partitioning is degenerate, recursion terminates and it returns the single table schema consisting of just $\mathcal{D}$. Otherwise, it recursively calls `Optimize` on each subtable to find the respective optimal snowflakes (the inner minimizations in Equation 3) and then creates a foreign key to link them, returning a snowflake decomposition of $\mathcal{D}$.

The large number of possible column partitions that must be explored in each call to `Optimize` makes a brute force enumeration intractable. Instead, we use branch-and-bound to constrain the search space. Our branch-and-bound search works by finding optimal solutions for increasingly large prefixes of $Cols(\mathcal{D})$. These prefix solutions are used to find upper and lower bounds for the optimal complete solution, which we use to prune the search space.

In `Optimize`, the search starts by initializing the two partitions, C_{parent} and C_{child} to the empty lists, and initializing C_{tail}, which contains the columns not yet explored by the search, to $Cols(\mathcal{D})$. At each node in the search we take the first column, c_{head}, from C_{tail} and branch, adding it to C_{parent} or C_{child} in two new nodes added to the search priority queue. When C_{tail} is empty, all columns have been partitioned, and the resulting partition gives the optimal solution. (Since the priority queue is ordered by our objective, any further solutions must be non-optimal.) At each node we also find the optimal solution for the prefix $\Pi^{+}_{C_{parent} \cup C_{child}}\mathcal{D}$. This is combined with loose bounds on the unexplored columns, $\Pi^{+}_{C_{tail}}\mathcal{D}$, to produce upper and lower bounds on the complete solution:



(a) Partition the columns of $\mathcal{D}$ into two subtables—$\mathcal{D}_{parent}$ and $\mathcal{D}_{child}$. Deduplicate the latter.

(b) Recursively find the optimal snowflake schema, $\mathcal{S}^*$, for each subtable (introducing foreign keys as needed).

(c) Create a new FK (blue) linking the root table of $\mathcal{S}^*_{parent}$ to the root of $\mathcal{S}^*_{child}$, creating a snowflake for $\mathcal{D}$.

Figure 4: SSLearn builds snowflake schemas through recursive binary partitioning of the input table. The recursive process terminates when no partitioning improves the objective, J. At each level in the recursion, branch-and-bound is used to find the partition which minimizes J, producing the optimal snowflake for $\mathcal{D}$.

Lower bound: Any optimal prefix solution provides a trivial lower bound on J for the complete solution. We can tighten this somewhat using the following theorem:

Theorem 4. *Suppose the objective J is of the form in Equation 6 with a canonical prior π described in Section 5.4. Let $\mathcal{S}^*, \mathcal{S}^*_C$ be the minimizers for the objectives $J(\cdot|\mathcal{D}), J(\cdot|\mathcal{D}_C)$, respectively. Then, for any $C \subset Cols(\mathcal{D})$:*

$$J(\mathcal{S}^*|\mathcal{D}) \quad \geq \quad J(\mathcal{S}^*_C|\mathcal{D}_C) + (1 - \tau_{tot}) \sum_{c \in C^C} |\Omega_c| \log_2 |\Omega_c|. \quad (10)$$

Upper bound: Any schema including all columns provides a trivial upper bound on the optimal solution. We can extend any prefix solution to include all columns by adding the unexplored columns to the root table, obtaining the following upper bound. Let $\mathcal{S}^*, \mathcal{S}^*_C$ be the minimizers for the objectives $J(\cdot|\mathcal{D}), J(\cdot|\mathcal{D}_C)$, respectively. Then, for any $C \subset Cols(\mathcal{D})$:

$$J(\mathcal{S}^*|\mathcal{D}) \leq J(\mathcal{S}^*_C|\mathcal{D}_C) + \sum_{c \in C^C} n \cdot H(c) \quad (11)$$

where $H(c)$ is the entropy of column c is the input data $\mathcal{D}$, and n is the number of rows in the root table. These bounds are inexpensive to evaluate since the column domains and entropies can be computed once and stored.

As `Optimize` proceeds—finding optimal solutions for increasingly long prefixes—these upper and lower bounds become tighter. Finding tight bounds early in the search can substantially improve the running time of the algorithm. To do this, we sort the columns from high to low entropy. By putting high entropy columns, which typically contain structurally important columns, early, and putting low entropy columns, which contribute little to the optimization objective, later, the search uncovers the structure of the optimal schema more quickly and tightens the upper and lower bounds.

Note that the same subproblems can occur many times in our recursive optimization. By Theorem 3, the solutions to these subproblems can be combined to produce the global solution, thus, we apply top-down dynamic programming, memoizing the results of `Optimize`. This greatly improves the efficiency of the algorithm.

Finally, our approach uses repeated binary partitioning of the same table to produce general n-ary trees. This introduces some redundancy since multiple partitioning sequences can produce the same final multi-way partition. Since this ordering does not affect the objective, we impose a canonical ordering using the minimum column index assigned to each partition.

6.2 Extension to an Anytime Algorithm

Despite the relative efficiency of the branch-and-bound plus dynamic programming algorithm described in the previous section, run time still grows exponentially as the number of columns grows, as shown in Figure 10. To address this, we show how to extend it to return a good approximate solution at any point in time. This has a second benefit of tightening the upper bound, which allows the search to prune more. Recall that previous two-part approaches must first find functional dependencies (the mostly costly part of the algorithm) before normalization, so can not provide early solutions.

The upper bound in Equation 11 corresponds to an approximate solution whose objective is inexpensive to compute but is likely a poor solution since all columns in C_{tail} are simply added to the root table. To find a better approximate solution, we occasionally run a more costly greedy allocation procedure that sequentially assigns the columns in C_{tail} to the table in the current schema which minimizes the objective J. The objective value of the resulting schema can be used as an improved upper bound.

Since this greedy allocation procedure is more expensive to run, it is triggered only when there is a possible change in the structure of the schema. We keep track of the partial schema associated with the upper bound from Equation 11 and run greedy allocation only when this upper bound changes in the root call to `Optimize`. If the algorithm reaches the time limit for the search before completion, it returns the best solution found so far which consists of the optimal solution for a prefix of the columns, plus greedy placement of the remaining columns.

Note that this greedy approach allocates the remaining columns to existing tables in the partial optimal schema; it does not create new tables or alter relationships in the schema. Thus, the exhaustive search component is necessary to find a globally optimal schema.

7 EXPERIMENTS

To evaluate the effectiveness of our proposed statistical schema learning (SSLearn) approach, we implemented a single-threaded version of our algorithm in Rust. We compare our results to the existing state-of-the-art by Papenbrock and Naumann (P&N) [30] which is parallelized and implemented in Java [28].

7.1 Experimental setup

All of our experiments for SSLearn were run on a dual CPU Intel Xeon E5-2630 v3 @ 2.40GHz machine with 16 physical cores and

192 GB of memory running CentOS Linux 7. For P&N, we ran some of the long running experiments on a faster dual CPU Intel Xeon Silver 4114 CPU @ 2.20GHz with 20 physical cores and 192 GB of memory. Since SSLearn is an anytime algorithm, we report the time to find the optimal solution (early termination at this point would get the optimal solution) and the total time to both find the solution and validate that it is optimal. P&N only returns a schema after completion, so we only report its total time.

As described in Section 5.4, our algorithm can leverage additional information, such as query histories, via informative priors. For a fair comparison with P&N, we do not take advantage of this; in the following experiments both algorithms use the same input data set. We do include a prior that prefers normalized tables with simple candidate keys; P&N's heuristic-based approach makes a similar assumption that effectively generates a heuristically constructed "golden set" of FDs. Concretely, we use the MDL objective (Equation 1), setting $\beta = 1$, and add a prior for tables with simple candidate keys. Following Example 3 in Section 5.4, we add a both a discouragement prior with $\gamma = 10$ to penalize tables without a simple candidate key and encouragement prior with $\tau = 1/2$ that encourages splitting out tables with a simple key. These choices have natural interpretation. The choice of γ penalizes a table without a simple candidate key by making it more expensive to store the foreign key. Each entry requires 10 more bits. The choice of τ halves the cost of storing the primary key when it is a simple key. We found that modifying the weight β on the MDL penalty had little effect on the learned schema as the evidence from the data, which grows linearly with the data size, dominated the prior, which does not. We found that, qualitatively, using both non-zero γ and τ had a significant impact on the quality of the learned schema. However, limited experiments when varying the parameter from $\gamma = 10$ to $\gamma = 15$ did not yield qualitatively different results. We did not explore varying τ.

Finally, to avoid the overhead of our greedy solution (Section 6.2) in simple cases where it doesn't provide much benefit, we only perform greedy allocation after the optimal schema on the first 15 columns has been computed.

7.2 Data

We evaluate our method on TPC-H, and a number of real-world data sets, including Musicbrainz [13] and 5 data sets from the CTU relational learning repository [25]. These data sets are already normalized which serves as the ground truth for good schema.

From each normalized data set, we construct denormalized tables involving some or all of the tables in the data set. In Table 1, we summarize these denormalized tables, indicating the source data set for each, the number of tables included, the total number of columns, and the shape of the join graph used for denormalizing the original schema: Star and Snowflake-shaped join graphs only include many-to-one relationships with a single root; Acyclic graphs include many-to-many joins, and Cyclic graphs include loops in the join graph. The Musicbrainz denormalized table is the same as that used in the prior state-of-the-art [30]; for this data set we know the source tables, but not the ground truth join graph.

Later, we also report our results on the North Carolina voter registration dataset [27] which is distributed as a single table, with no ground truth schema. This dataset has 8.2M rows and 71 columns.

7.3 Quality of Learned Schemas

To evaluate the quality of the learned schemas, we compare them to the ground truth normalized schemas. Since we are not aware of an existing schema similarity metric appropriate for this task, we use an *edit distance* which measures the number of editing operations a person might perform to make the learned schema match the tables in the ground truth schema. To define the edit distance, we consider three editing operations on schemas: (1) merging two tables, (2) splitting a table into two, and (3) moving one column between two tables. The edit distance is the minimum number of such edit operations on a learned schema to match the original schema. We say a learned schema matches an original schema if the entities in the original schema have the same data columns (as opposed to FK columns) as entities in the learned schema. More precisely, the schemas match if there is an injective mapping from the tables in the learned schema to the original schema and for every learned table t with corresponding table t_0 in the original schema, t columns are a subset of t_0's. While there are lines of work on automatically computing various tree edit distances [8, 32], we could not find existing algorithms that compute the edit distance we defined or one with similar semantics. Instead, we computed the edit distance by hand. This was possible in our experiments since the distances were small and mostly involved easy to resolve merge operations. Learned schemas and code are available at [3].

Since our main task is to identify meaningful entities in the data but a number of our datasets are generated from non-snowflake schemas, we do not compare the graphical structure of the learned and the original schema; we only compare the entities themselves.

Since P&N generates logical schemas with primary keys, we convert P&N schemas to conceptual schemas by erasing the FK information as shown in Figure 2. In some cases, P&N may duplicate a column to form primary keys in multiple tables; for example, in Figure 6, `c_id` appears in multiple tables corresponding to the true `Track` and `Recording` tables. In these cases, we assign the column to the table that casts P&N in the best light, namely the table that minimizes the edit distance to the underlying schema.

We also break down the edit distance by number of edit operations of each type. Since there are multiple edit sequences that can convert from one schema to another, we use edit sequences that prefer merges to column moves and column moves moves to splits. This preference favors operations that require fewer choices. Choosing two tables to merge is considered easier than choosing what subset of columns to split out. The former has at most quadratically many choices while the latter has exponentially many.

Assignment errors are ignored for duplicate columns in the denormalized table when they belong to different tables in the original schema (such as FK-PK pairs); no solely data-driven algorithm can distinguish these columns. Tables in the original schema joined by one-to-one relationships are also considered a single table as they cannot be distinguished using only their distribution of values.

7.4 Results on Quality

Figure 5 shows the number of edits of each type and the overall edit distance across all data sets for SSLearn (blue) and P&N (red). SSLearn learns schemas that are much closer to the originals, with a total edit distance that is $1/5^{th}$ of P&N's. P&N is highly susceptible

	Input			Number of tables in learned schema and edit distance to original schema									
				SSLearn					P&N				
Dataset	Tbls	Cols	Schema	Tbls	Merge	Split	Move	Distance	Tbls	Merge	Split	Move	Distance
TPC-H	8	52	Snowflake	10	2	0	0	***2***	15	7	0	1	8
Musicbrainz	11	113	Unknown	11	0	0	2	***2***	20	10	1	8	19
PKDD Financial	6	45	Acyclic	5	0	1	0	***1***	9	4	2	0	6
MovieLens (actors)	5	13	Acyclic	3	0	0	2	2	5	0	0	1	***1***
MovieLens (users)	5	14	Acyclic	4	0	0	1	1	5	0	0	1	1
FNHK	3	22	Cyclic	5	1	0	0	***1***	10	5	0	0	5
StackOverflow (posts)	4	44	Snowflake	4	0	0	0	***0***	25	18	0	0	18
StackOverflow (votes)	5	48	Acyclic	8	1	0	0	***1***	10	5	0	0	5
CCS Transactions	4	18	Star	5	1	0	1	***2***	8	4	0	0	4

Table 1: Quality of learned schemas. SSLearn generates schemas much closer to the ground truth as measured by the edit distance.

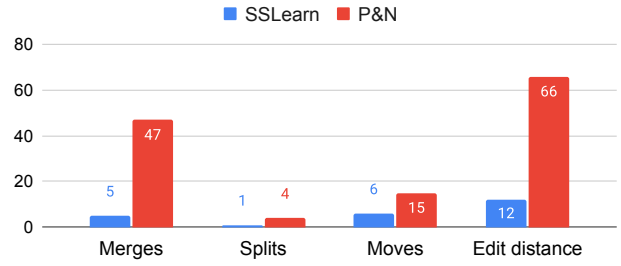


Figure 5: Number of individual edits and total edit distance from original normalized schema for the data sets in Table 1 (lower is better).

to over-splitting, producing many more tables than in the original schemas. For example, the original schema for TPC-H has 8 tables. SSLearn learns a schema with 10 tables; merging 2 pairs of these tables reconstructs the original schema. In contrast, P&N learns a schema with 15 tables; merging 7 pairs of tables and moving one column between tables is required to reconstruct the original schema. This is broken down in Table 1, showing that SSLearn's advantage holds even if the underlying schema does not satisfy the model assumption of a hierarchical snowflake. P&N returns a marginally higher quality result in only 1 out of 9 cases.

Furthermore, we note that in a number of cases labeled as an error, SSLearn uncovers a meaningful real world entity. For example, the CCS dataset consists of gas station transactions. SSLearn generates a semantically reasonable table identifying the gas station chain. However, this is not modeled in the original `gasstations` schema, so we classify this as an error. Similarly, in TPC-H, SSLearn splits out the Part Manufacturer from the Parts table—a semantically reasonable normalization that does not exist in the original schema—and it uses an additional table to model a cross-column dependency in the definition of TPC-H where all line items with `linestatus='O'` have a ship date before `1995-06-18` and all with `linestatus='F'` have ship date on or after `1995-06-18`. In contrast, Figures 6 and 8 illustrate that the over-splitting behavior of P&N often does not produce semantically meaningful entities.

We note that for non-snowflake schemas, such as in the StackOverflow (votes) dataset, both SSLearn and P&N learn junction tables that do not exist in the original schema due to the snowflake schema constraint of both algorithms. Even in these cases, however, SSLearn learns the entities in the source schema.

To qualitatively demonstrate the difference in learned schema quality, the left side of Figure 6 shows the results for the MusicBrainz data set from Papenbrock and Naumann [30] compared to the results of SSLearn on the right. Each node in the tree represents a table in the learned schema. To simplify the display, we use `l_*` to denote a set of column names with prefix `"l_"`.

In both cases, the methods learn a snowflake schema with a root junction table to capture the many-to-many relationships among Artist, Place, Release label, and Track that exist in the original schema. However, in this case, P&N greatly over partitions the schema, producing 20 tables, compared to the 11 tables in the original schema (extra tables are marked with a red triangle; these need to be merged into a parent table to reconstruct the original schema). It also fails to correctly place a large number of low cardinality columns (highlighted in yellow; these need to be moved to their correct location to reconstruct the original schema). In contrast, SSLearn yields a near perfect snowflake; only two very low cardinality columns are misplaced. Furthermore, although these columns, `a_type` and `a_edits_pending`, belong to the `artist` table, they are functionally dependent on the primary key `ac_id` of the smaller `artist_credit` table. We also note that the two methods place `artist_credit` under two different tables. In this case, both placements can be considered correct since there is a FK-PK relationship between both `artist` and `artist_credit` as well as `release_group` and `artist_credit`, however, SSLearn associates it with the smaller `artist` table which is arguably a more semantically meaningful relationship.

7.5 Impact of noise

To test the robustness of the algorithms to noise in the dataset, we took the TPC-H data set and randomly replaced 0.02% of the entries with a randomly chosen value from the same column. This corresponds to corrupting 1% of rows. This more challenging case required significantly more time so we used only 20% of orders from the denormalized TPC-H dataset.

The resulting schemas learned for this noisy version of TPC-H are shown in Figure 7. Despite the noise, SSLearn is still able to

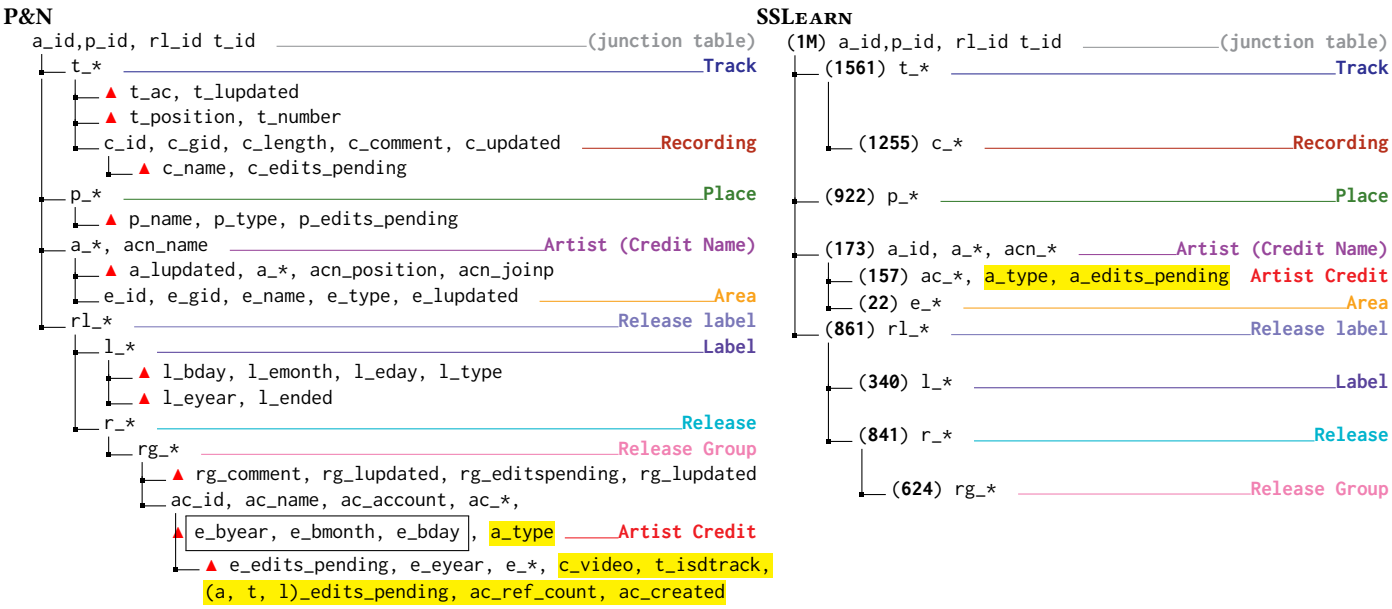


Figure 6: Comparison of learned schemas on MusicBrainz. The `prefix_*` wildcard replaces multiple columns of that form, and `(a,b)_c` denotes `a_c, b_c`. We annotate the edits needed to transform the learned schemas into the ground truth: boxed entities should be further split, entities with a ▲ should be merged with a parent entity, and highlighted columns should be moved to a different branch in the schema. P&N heavily over-partitions the schema and misplaces low cardinality columns due to semantically unimportant FDs in the data. In contrast, SSLearn very accurately learns the original schema, only misplacing two very low cardinality columns.

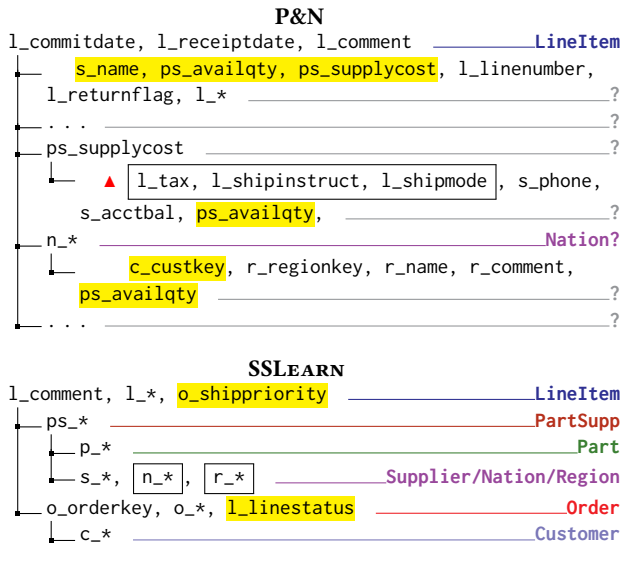


Figure 7: Schemas learned from TPC-H with noise. *Top:* 7 of the 23 tables generated by P&N are shown. Few map to semantically meaningful entities. *Bottom:* SSLearn recovers the ground truth TPC-H schema almost exactly; two columns are misplaced and Suppliers, Nations, and Regions are placed in a single table.

recover most of the structure in TPC-H. It collapses the Supplier, Nation, and Region tables into one table, and it assigns `o_shippriority` to the LineItems table and `l_linestatus` to Orders. However, the core structure is successfully identified.

In contrast, we could identify little meaningful structure in the schema learned by P&N, In this case, P&N generates 23 tables. However, unlike the noise-free case where the additional tables can simply be merged together to reconstruct the original schema, in this noisy scenario, many tables are composed of attributes drawn from multiple tables across the original schema. Figure 7 shows portions of the schema learned by P&N; no straightforward transformation of this schema will recover the ground truth. Thus, P&N was unable to uncover semantically meaningful structure in the presence of a small amount of noise.

7.6 Real world denormalized dataset

We also demonstrate SSLearn on the North Carolina voter registration dataset which is distributed as a single table and does not have a ground truth schema. Figure 8 shows the first few levels of the schemas for P&N and SSLearn. P&N learns many very high cardinality tables and scatters voter demographic columns across the schema due to incidental functional dependencies. In constrast, SSLearn arranges the columns into a concise, semantically interpretable schema with a root table containing voter specific attributes, a hierarchy of increasingly specific geographic regions, and set of low cardinality dictionary tables.

7.7 Performance

In addition to finding higher quality schemas, Figure 9 shows that SSLearn is significantly faster than P&N, by up to two orders of

P&N

```
(1M) ncid, last_name, first_name, middle_name
  (999,999) mail_addr(1-2), mail_(city, zipcode),
  full_phone_number, name_suffix_lbl, birth_state, registr_date
    (974,392) res_street_addr, race_code, birth_age,
    reason_cd, precinct_(abbrv, desc), vtd_(abbrv, desc)
    (975,307) res_street_addr, mail_addr(3-4), gender_code
    . . .
    (916,478) res_street_addr, drivers_lic, municipality_*
      (906,019) ethnic_code, party_cd, cong_dist, super_court,
      judic_dist, nc_senate, nc_house, . . .
        (855,117) res_street_addr, ward_*,
        (sewer, sanit, rescue)_dist_*
          . . .
    (994,150) voter_reg_num, birth_year
      . . .
  (16) county_(id, desc)
```

SSLearn

```
(1M) ncid, voter_reg_num, phone_num,
res_street_address, mail_addr1,
(first,middle,last)_name,
birth_(age,year,state),
(party, gender, race, ethnic)_code, drivers_lic

  (50082) mail_addr(2-4), mail_(zipcode, city, state),
  name_suffix_lbl, fire_district
    (1628) zip_code, res_city_desc, state_cd,
    (water,sewer)_dist
    (township, ward)_(abbrv,desc), confidential_ind
      (542) (vtd, precinct, sanit_dist)_(abbr, desc)
        . . .
    (5) status_cd, voter_status_desc
    (27) reason_cd, voter_status_reason_desc
```

Figure 8: Learned schemas on the NC voter registration dataset. `(a,b)_(c,d)` denotes the 4 attributes: `a_c, b_c, a_d, b_d`. P&N over-partitions tables and scatters demographic information throughout the schema. SSLearn produces a concise, semantically meaningful schema with voter demographic information in the root table and a geographic hierarchy.

	Input			SSLearn		P&N	Speedup	
Dataset	Rows	Cols	Schema	Solution time (min)	Total time	Total time	Solution	Total
CCS Transactions	1 K	18	Star	0.00	0.00	0.01	30.00	30.00
MovieLens (actors)	148 K	13	Acyclic	0.01	0.01	0.03	6.57	6.57
MovieLens (users)	1.1 M	14	Acyclic	0.03	0.03	0.18	6.89	6.89
FNHK	2.3 M	22	Cyclic	0.80	0.90	1.88	2.35	2.09
StackOverflow (posts)	213 K	44	Snowflake	3.80	4.00	12.37	3.25	3.09
PKDD Financial	1.3M	45	Acyclic	1.20	1.70	12.32	10.26	7.25
StackOverflow (votes)	3.1 M	48	Acyclic	1.00	1.70	226.35	226.35	133.15
TPC-H (sampled + noisy)	1.1 M	52	Snowflake	2.40	132.90	364.78	151.99	2.74
NC Voter	1 M	71	Unknown	33.18	33.54	547.80	16.51	16.33
Musicbrainz	1 M	113	Unknown	2.50	1125.88	1775.73	710.29	1.58
TPC-H	6 M	52	Snowflake	20.00	24.00	1973.77	98.69	82.24

Table 2: s. SSLearn is faster for all cases we tried, by up to two orders of magnitude (Total Speedup column).

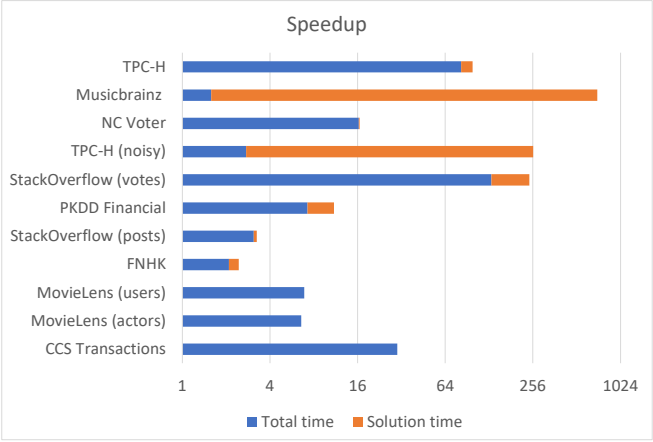


Figure 9: Speedup $\left(\frac{\text{P\&N runtime}}{\text{SSLearn runtime}}\right)$ by dataset. Blue bars use the total time spent by SSLearn while orange bars use the time at which SSLearn found its solution.

magnitude on large datasets. Our single-threaded algorithm exhaustively searches through all 52 columns of TPC-H in 25 minutes while P&N requires over 1.5 days despite running on 20 physical cores. SSLearn was faster on all data sets we tested. Detailed performance results are shown in Table 2.

Furthermore, although the search may take a long time, the optimal solution may be found much more quickly. In some cases, much of the time is spent validating that an early candidate solution is actually the optimal solution. For the MusicBrainz dataset, the final solution is found at 2.5 minutes and 19 hours is spent verifying that the solution is optimal. This property also allows us to build methods that execute in bounded time. Since SSLearn found the solution within 35 minutes for all our datasets, capping the runtime at 1 hour would yield a procedure that executes in bounded time and returns the optimal result. However, the long stretches of time where the solver is verifying if the current solution is optimal make it difficult to predict what is an appropriate cutoff time to find a good schema. We found that the time to find the optimal solution depended heavily on the order in which columns are searched. If,

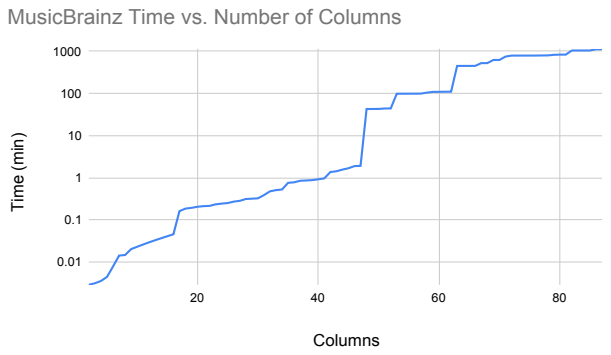


Figure 10: Time to learn best schema on first x columns.

for every table, some candidate key appeared early in the search, then the solution was found very quickly.

Even though SSLearn is much faster than P&N, Figure 10 demonstrates on the MusicBrainz data set that our algorithm's run time is still exponential in the number of columns processed in the exhaustive search. Thus, the ability to maintain good candidate solutions at all times is a useful property of the algorithm.

8 ANALYSIS

The statistical framework developed in Section 5 allows us to analyze how our method works. In particular, it allows us to examine when our method chooses to split a table, and the role of the column independence assumption in our generative model. This helps us understand the unique properties and advantages of our method versus related information theoretic work.

Consider a schema $\mathcal{S}$ with table t. For a set of columns $C \subset Cols(t)$, let $\mathcal{S}'$ be the schema obtained by splitting out the columns C from t to form a deduplicated table t'.

Theorem 5. *Consider schemas $\mathcal{S}, \mathcal{S}'$ where the only difference is that for some table $t \in Tbls(\mathcal{S})$ and columns $C \subset Cols(t)$, $\mathcal{S}'$ splits t and puts columns C into a new table t'. Consider tuples from columns C in t, and denote the empirical joint distribution of tuples by $\hat{p}_{t'}$. Schema learning prefers the split schema $\mathcal{S}'$ if and only if*

$$n_t \, KL\left(\hat{p}_{t'} \,\middle\|\, \prod_{c \in C} \hat{p}_c\right) > n_{t'} \sum_{c \in Cols(t')} H(c) = Size(t') \qquad (12)$$

where $\hat{p}_{t'}, \hat{p}_c$ are the respective empirical distributions for rows in t' or column c. KL is the Kullbeck-Leibler divergence measuring the difference in distributions.

The l.h.s. represents the space savings from replacing redundant information in multiple columns in the original table with a foreign key. The r.h.s. represents the cost of creating a new child table.

8.1 Comparison to Approximate FD measure

We can compare this to an information theoretic measure for detecting approximate FDs. The fraction of information measure [23] is used to test if the FD $X \rightarrow Y$ exists, and it is defined to be

$$F(X;Y) := \frac{H(Y) - H(Y|X)}{H(Y)} = \frac{KL(\hat{p}_{XY} || \hat{p}_X \hat{p}_Y)}{H(Y)} \qquad (13)$$

where $\hat{p}_Y, \hat{p}_{XY}$ respectively denote the empirical distribution of all columns in Y and all columns in X and Y. It rejects the null hypothesis that $X \rightarrow Y$ is an FD if $F(X;Y) > c$ for some constant c or equivalently if $KL(\hat{p}_{XY} || \hat{p}_X \hat{p}_Y) > cH(Y)$. If the FD is detected, then an FD driven schema normalization method may split the columns in X, Y into another table t'.

Comparing equations 12 and 13, the main similarity is the use of the KL-divergence to measure dependence. It compares a model where the joint distribution ($\hat{p}_{t'}$ or $\hat{p}_{XY}$) of columns in t' is fully dependent with one which treats columns or groups of columns as independent. This also justifies the column independence assumptions in our model. Like with approximate FD measures, independence assumptions provide a useful measure of dependence even if the columns are not truly independent.

Our measure has a few key differences and advantages as well. First, Equation 12 incorporates the size of the tables n_t, n'_t. This key difference makes our method sensitive to the duplication that naturally occurs in many-to-one joins. In comparison, entropic measures that rely only on the *distribution* of values can only indirectly measure the amount of duplication. Thus, rather than using a generic measure of dependence, our objective is specifically suited for detecting joins. Secondly, our method is easier to apply. It does not require choosing thresholds or how much dependence is appropriate. Thirdly, our method does not require additional bias correction to avoid excessive false positives, unlike the approximate FD measure [23].

9 DISCUSSION AND FUTURE WORK

Our schema learning method has obvious application in multiple applications, including in data analysis and exploration. By finding meaningful clusters and hierarchies of discrete valued columns, it can help data analysts to better understand the relationships between columns during data exploration. Extending our model to handle continuous valued columns is a potential area of future work. A second possible direction for future work is extend our method to learn acyclic schemas. Our experiments in section 7.4 identify several cases where our method learns the meaningful entities in the schema but create junction tables where there are many-to-many relationships. Removing these junction tables as a post-processing step could recover the original acyclic schema. Another future direction is to extend our approach to explicitly model noise in the data, for example by using a noise model to cluster similar rows. This would increase the robustness of Statistical Schema Learning.

A direction for future algorithmic work is to use Markov Chain Monte Carlo (MCMC) methods. This would allow for even more flexible priors that do not need to have a recursive decomposition as well as allow it to better explore allocations of columns that are searched near the end of an exhaustive search. However, it would lose efficiencies obtained by dynamic programming.

Statistical Schema Learning does not explicitly force the learned schema to follow a canonical normal form. However, existing methods that guarantee normalization into a canonical normal form can be run as a post-processing step. Our approach can accelerate those methods as the normalization can be performed on smaller subtables with fewer columns than the original table.

10 CONCLUSION

We propose a new paradigm for schema learning, driven by directly optimizing for good properties, rather than assuming that conforming to a canonical normal form will naturally yield a good schema. Chief among these is the property of minimum description length which quantifies Occam's razor. Thus, even as an unsupervised learning method without a ground truth to compare to, our model has a principled method for picking out meaningful real world entities from denormalized data sets. By developing a Bayesian statistical framework, the method is made extensible, allowing other desirable properties to be incorporated into the optimization criterion. Importantly, we make computation under this paradigm tractable, by developing an efficient branch-and-bound algorithm with dynamic programming. Our experiments show that this combination of statistical modeling plus an efficient optimization algorithm produces results that are of significantly higher quality and more robust to noise in much less time than existing methods.

A PROOFS

Theorem 1 proof. We show that even the simpler class of star schemas is already superexponential. Consider the number of ways to partition m columns. This is the Bell number B_m which asymptotically satisfies $\log B_m = n \log n + O(n \log \log n)$. □

Theorem 2 proof. We start by showing the empirical entropy is the negative maximum log-likelihood of a $Multinomial(n, p_1, \dots p_k)$. Let X be a draw from this distribution. The MLE of the class probabilities is trivially the empirical probabilities $\hat{p}_k = X_k/n$. The maximum log-likelihood in base 2 is $\ell(\hat{p}|\mathcal{D}) = -\sum_k n \frac{x_k}{n} \log_2 \hat{p} = -nH(X)$.

Each column is independent and distributed as a multinomial distribution. Since columns are generated independently, the maximum log-likelihood is

$$\sup_{\rho} \ell(\mathcal{S}, \rho|\mathcal{D}) = \sum_{t \in Tbls(\mathcal{S})} \sum_{c \in Cols(t)} -n_t H(c) = -\sum_{t \in Tbls(\mathcal{S})} Length(t)$$

□

Theorem 3 proof. Let C be the columns assigned to some child t and its descendents of the root in $\mathcal{S}$. Let $\mathcal{S}_t$ denote the subtree rooted at t and let $\mathcal{S}_0$ be $\mathcal{S}$ with $\mathcal{S}_t$ erased. Simple algebra yields

$$J(\mathcal{S}|\mathcal{D}) = w(\mathcal{D}_{C^C}, \mathcal{D}^U_C) + nH(c_{fk}) + g(\mathcal{D}, \tilde{\mathcal{D}}_t) + J\left(\mathcal{S}_0 \mid \mathcal{D}_{C^C}\right) + J\left(\mathcal{S}_t \mid \mathcal{D}^U_C\right)$$

where c_{fk} is the column that acts as a FK to t. We can take $w(\mathcal{D}_{C^C}, \mathcal{D}^U_C) = nH(c_{fk}) + g(\mathcal{D}, \tilde{\mathcal{D}}_t)$ where c_{fk} is the foreign key column joining the root to table t. Minimizing this objective gives

$$\begin{aligned}\min_{\mathcal{S}} J(\mathcal{S}|\mathcal{D}) &= \min_{C \subset Cols(\mathcal{D})} \min_{S_0, S_1} \Big(w(\mathcal{D}_{C^C}, \mathcal{D}^U_C) + J\left(S_0 \mid \mathcal{D}_{C^C}\right) + J\left(S_1 \mid \mathcal{D}^U_C\right) \Big) \\ &= \min_{C \subset Cols(\mathcal{D})} \Big(w(\mathcal{D}_{C^C}, \mathcal{D}^U_C) + \min_{S_0} J\left(S_0 \mid \mathcal{D}_{C^C}\right) + \min_{S_1} J\left(S_1 \mid \mathcal{D}^U_C\right) \Big)\end{aligned}$$

□

The last equality follows since w is constant given C. Changes to $\mathcal{S}_0$ do not affect $\mathcal{S}_1$ and vice versa since their columns are disjoint.

Theorem 4 proof. Consider the minimizer $\mathcal{S}^*$. Let $\mathcal{T}_1$ denote the set of tables that only contain columns from C^C and $\mathcal{T}_1$ denote the remaining tables that contain some column of C^C. The objective can be written as the sum of table costs. We first examine the cost of tables in $\mathcal{T}_1$. For $t \in \mathcal{T}_1$,

$$\begin{aligned}n_t \sum_{c \in Tbls(t)} H(c) - \tau_{tot} n_t \log_2 n_t &\geq (1 - \tau_{tot}) n_t \sum_{c \in Tbls(t)} H(c) \\ &\geq (1 - \tau_{tot}) n_t \sum_{c \in Tbls(t)} |\Omega_i| \log_2 |\Omega_i|.\end{aligned}$$

The first inequality follows from the fact that the entropy of a joint distribution is at most the sum of marginal entropies.

For a table t in $\mathcal{T}_0$, denote by t_0 the table before any columns in C^C were added and n_0 the number of rows in t_0. We examine the case where there is only one added column c from C^C. Including an additional column can only add additional rows, so we split the table into 3 parts: (1) a subtable of t that is equal to t_0, consisting of a subset of rows and all columns except c, (2) the remaining rows and all columns except c, (3) the values in column c. The cost of coding t with one code is lower bounded by the cost of coding all 3 parts separately with an optimal code for each of the three parts. The cost of coding part (1) using an optimal columnwise coding is just the original cost of coding t_0. The change in the prior after adding c is $\Delta_\pi = \tau_{tot}(n_0 \log_2 n_0 - n_t \log_2 n_t)$. Since the function $x \log_2 x$ is convex, Jensen's inequality gives that $\mathbb{E}Z \log_2 Z > (\mathbb{E}Z) \log_2 \mathbb{E}Z$. Taking Z to be n_0 or $n_t - n_0$ with equal probability, one finds that $\frac{n_t}{2} \log_2 \frac{n_t}{2} < \frac{n_t - n_0}{2} \log_2(n_t - n_0) + \frac{n_0}{2} \log_2(n_0)$ and equivalently, $n_0 \log_2 n_0 - n_t \log_2 n_t > -(n_t - n_0) \log_2(n_t - n_0) - n_t$. Thus $\Delta_\pi > -\tau_{tot}(n_t - n_0) \log_2(n_t - n_0)$. The cost of coding (2) and (3) separately is greater than or equal to the cost of coding them using their joint distribution, and there are at least $v = \max\{n_t - n_0, |\Omega_c|\}$ distinct values, their combined cost must be at least $v \log_2 v$. Thus, the change in the objective must be at least $v \log_2 v + \Delta_\pi > v \log_2 v - \tau_{tot} v \log_2 v > (1 - \tau_{tot}) |\Omega_c| \log_2 |\Omega_c|$.

□

Theorem 5. A split is prefered if $J_{MLE}(\mathcal{S}'|\mathcal{D}) - J_{MLE}(\mathcal{S}|\mathcal{D})$. Columns in C^C contribute the same amount to the log-likelihood in $\mathcal{S}$ and $\mathcal{S}'$, so they can be ignored. The table t' that is split out only contributes to the objective of $\mathcal{S}'$ and is $Length(t')$. This only leaves a comparison between the added foreign key in $\mathcal{S}'$ and the contribution of columns C in $\mathcal{S}$. In $\mathcal{S}'$ the distribution of the foreign key is the empirical joint distribution for columns C. Expanding the difference in log-likelihoods gives

$$\begin{aligned}&J_{MLE}(\mathcal{S}'|\mathcal{D}) - J_{MLE}(\mathcal{S}|\mathcal{D}) \\ &= n_t \sum_x \hat{p}_C(x) \left(\log \hat{p}_C(x) - \sum_{c \in C} \log \hat{p}_c(x) \right) - Length(t') \\ &= n_t \sum_x \log \hat{p}_C(x) \left(\hat{p}_C(x) - \log \prod_{c \in C} \hat{p}_c(x) \right) - Length(t') \\ &= KL\left(\hat{p}_C \middle\| \prod_{c \in C} \hat{p}_c \right) - Length(t').\end{aligned}$$

□

REFERENCES

[1] Ziawasch Abedjan, Patrick Schulze, and Felix Naumann. 2014. DFD: Efficient Functional Dependency Discovery. In *CIKM 2014*. 949–958.

[2] Periklis Andritsos, Renée Miller, and Panayiotis Tsaparas. 2004. Information-Theoretic Tools for Mining Database Structure from Large Data Sets.. In *SIGMOD*. 731–742.

[3] Anonymous. 2022. Schema Learning. https://anonymous.4open.science/r/schema_learning-EC93/README.md.

[4] Catriel Beeri and Philip A. Bernstein. 1979. Computational Problems Related to the Design of Normal Form Relational Schemas. *ACM Trans. Database Syst.* 4, 1 (March 1979), 30–59. https://doi.org/10.1145/320064.320066

[5] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. 1983. On the Desirability of Acyclic Database Schemes. *J. ACM* 30, 3 (July 1983), 479–513. https://doi.org/10.1145/2402.322389

[6] Philip A. Bernstein. 1976. Synthesizing Third Normal Form Relations from Functional Dependencies. *ACM Trans. Database Syst.* 1, 4 (Dec. 1976), 277–298. https://doi.org/10.1145/320493.320489

[7] Laure Berti-Équille, Hazar Harmouch, Felix Naumann, Noël Novelli, and Saravanan Thirumuruganathan. 2018. Discovery of Genuine Functional Dependencies from Relational Data with Missing Values. *Proc. VLDB Endow.* 11, 8 (April 2018), 880–892. https://doi.org/10.14778/3204028.3204032

[8] Philip Bille. 2005. A survey on tree edit distance and related problems. *Theoretical computer science* 337, 1-3 (2005), 217–239.

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

[10] Jim Diederich and Jack Milton. 1988. New Methods and Fast Algorithms for Database Normalization. *ACM Trans. Database Syst.* 13, 3 (Sept. 1988), 339–365. https://doi.org/10.1145/44498.44499

[11] Mathias Drton and Marloes H Maathuis. 2017. Structure Learning in Graphical Modeling. *Annual Review of Statistics and Its Application* 4, 1 (2017), 365–393.

[12] Peter A. Flach and Iztok Savnik. 1999. Database dependency discovery: a machine learning approach. *Ai Communications* 12, 3 (1999), 139–160.

[13] Musicbrainz foundation. 2020. Musicbrainz. data retrieved from , https://musicbrainz.org/.

[14] Yihan Gao and Aditya Parameswaran. 2016. Squish: Near-Optimal Compression for Archival of Relational Datasets. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 1575–1584. https://doi.org/10.1145/2939672.2939867

[15] Bogdan Ghita, Diego G. Tomé, and Peter A. Boncz. 2020. White-box Compression: Learning and Exploiting Compact Table Representations. In *CIDR 2020*. http://cidrdb.org/cidr2020/papers/p4-ghita-cidr20.pdf

[16] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. Tane: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *Comput. J.* 42, 2 (1999), 100–111.

[17] Amir Ilkhechi, Andrew Crotty, Alex Galakatos, Yicong Mao, Grace Fan, Xiran Shi, and Ugur Cetintemel. 2020. DeepSqueeze: Deep Semantic Compression for Tabular Data. In *SIGMOD 2020*. Association for Computing Machinery, New York, NY, USA, 1733–1746.

[18] Ali Jalali, Christopher C Johnson, and Pradeep Ravikumar. 2011. On Learning Discrete Graphical Models using Greedy Methods. In *NIPS*.

[19] Lan Jiang and Felix Naumann. 2020. Holistic primary key and foreign key detection. *Journal of Intelligent Information Systems* 54 (06 2020), 1–23. https://doi.org/10.1007/s10844-019-00562-z

[20] Batya Kenig, Pranay Mundra, Guna Prasaad, Babak Salimi, and Dan Suciu. 2020. Mining Approximate Acyclic Schemes from Relations. In *SIGMOD 2020* (Portland, OR, USA). New York, NY, USA, 297–312. https://doi.org/10.1145/3318464.3380573

[21] Jyrki Kivinen and Heikki Mannila. 1995. Approximate inference of functional dependencies from relations. *Theoretical Computer Science* 149, 1 (1995), 129–149.

[22] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. 2000. Efficient Discovery of Functional Dependencies and Armstrong Relations. In *EDBT 2000*, Vol. 1777. 350–364.

[23] Panagiotis Mandros, Mario Boley, and Jilles Vreeken. 2017. Discovering Reliable Approximate Functional Dependencies. In *KDD 2017*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3097983.3098062

[24] Panagiotis Mandros, Mario Boley, and Jilles Vreeken. 2018. Discovering reliable dependencies from data: Hardness and improved algorithms. In *2018 IEEE international conference on data mining (ICDM)*. IEEE, 317–326.

[25] Jan Motl and Oliver Schulte. 2015. The CTU Prague Relational Learning Repository. *CoRR* (2015). http://arxiv.org/abs/1511.03086

[26] Noel Novelli and Rosine Cicchetti. 2001. FUN: An Efficient Algorithm for Mining Functional and Embedded Dependencies. In *ICDT '01 Proceedings of the 8th International Conference on Database Theory*. 189–203.

[27] North Carolina Board of Elections. 2020. Voter registration. data retrieved from , https://www.ncsbe.gov/results-data/voter-registration-data.

[28] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. 2015. Data Profiling with Metanome. *Proc. VLDB Endow.* 8, 12 (Aug. 2015), 1860–1863. https://doi.org/10.14778/2824032.2824086

[29] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In *SIGMOD 2016 (SIGMOD '16)*. New York, NY, USA.

[30] Thorsten Papenbrock and Felix Naumann. 2017. Data-driven Schema Normalization. In *EDBT 2017*.

[31] D. S. Pavlichin, A. Ingber, and T. Weissman. 2017. Compressing Tabular Data via Pairwise Dependencies. In *2017 Data Compression Conference (DCC)*. 455–455. https://doi.org/10.1109/DCC.2017.82

[32] Mateusz Pawlik and Nikolaus Augsten. 2015. Efficient computation of the tree edit distance. *ACM Transactions on Database Systems (TODS)* 40, 1 (2015), 1–40.

[33] Christian P Robert et al. 2007. *The Bayesian choice: from decision-theoretic foundations to computational implementation*. Vol. 2. Springer.

[34] Catharine Wyss, Chris Giannella, and Edward L. Robertson. 2001. FastFDs: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances - Extended Abstract. In *DaWaK International Conference on Data Warehousing and Knowledge Discovery*. 101–110.

[35] Hong Yao, H.J. Hamilton, and C.J. Butz. 2002. FD/spl I.bar/Mine: discovering functional dependencies in a database using equivalences. In *2002 IEEE International Conference on Data Mining*. 729–732.

[36] Yunjia Zhang, Zhihan Guo, and Theodoros Rekatsinas. 2020. A Statistical Perspective on Discovering Functional Dependencies in Noisy Data. In *SIGMOD 2020*.