On Multi-Column Foreign Key Discovery

Meihui Zhang
National University of Singapore
mhzhang@comp.nus.edu.sg

Marios Hadjieleftheriou
AT&T Labs - Research
marioh@research.att.com

Beng Chin Ooi
National University of Singapore
ooibc@comp.nus.edu.sg

Cecilia M. Procopiuc
AT&T Labs - Research
magda@research.att.com

Divesh Srivastava
AT&T Labs - Research
divesh@research.att.com

ABSTRACT

A foreign/primary key relationship between relational tables is one of the most important constraints in a database. From a data analysis perspective, discovering foreign keys is a crucial step in understanding and working with the data. Nevertheless, more often than not, foreign key constraints are not specified in the data, for various reasons; e.g., some associations are not known to designers but are inherent in the data, while others become invalid due to data inconsistencies. This work proposes a robust algorithm for discovering single-column and multi-column foreign keys. Previous work concentrated mostly on discovering single-column foreign keys using a variety of rules, like inclusion dependencies, column names, and minimum/maximum values. We first propose a general rule, termed *Randomness*, that subsumes a variety of other rules. We then develop efficient approximation algorithms for evaluating randomness, using only two passes over the data. Finally, we validate our approach via extensive experiments using real and synthetic datasets.

1. INTRODUCTION

A foreign/primary key relationship between relational tables is one of the most important constraints in a database. From a data analysis perspective, discovering foreign keys is a crucial step in understanding and working with the data. For that reason, database systems allow the explicit specification of foreign key constraints in the database schema. Nevertheless, in practice, database designers frequently fail to specify such constraints for various reasons, including: they are not aware of implicit relationships inherent in the data; such relationships might hold across multiple databases; it is not feasible to specify the constraints due to data inconsistencies (e.g., those arising from data integration or from database evolution over time); or because of performance considerations. When this happens in enterprise databases, which often contain hundreds of tables, thousands of columns and insufficient (or missing) documentation, even expert users have a difficult time identifying foreign key constraints.

In this paper, we propose a novel approach for discovering foreign/primary key (fk/pk) relationships between single or multiple columns in relational databases. Surprisingly, little previous work deals with the case of discovering multi-column foreign keys [14]. Even for single-column keys, existing work is limited and focuses mainly on identifying inclusion dependencies, since the only formal requirement for specifying a foreign key constraint is that the foreign key be a subset of the primary key [14,1]. However, checking only for inclusion can lead to a large number of false positives.

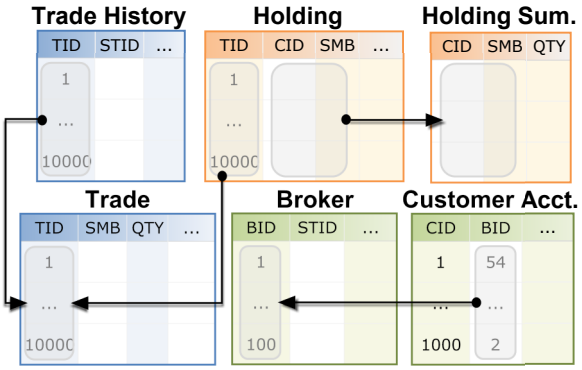


Figure 1: A small subset of the TPC-E schema with one multi-column and several single-column foreign keys.

For example, Figure 1 shows a portion of the benchmark TPC-E schema, which represents a stock transaction system. It has information about customer accounts, companies, brokers, stock trades, etc. Column Trade.TID contains all integers in the interval $[1, 10000]$, while column Broker.BID, which is unrelated to TID, contains all integers in $[1, 100]$. A simple inclusion test would incorrectly report (Broker.BID, Trade.TID) as a foreign/primary key pair. This scenario arises frequently in practice because of auto-increment fields. Of course, one could adapt the test so that it discards pairs in which one column is a consecutive subset (e.g., a prefix or a suffix) of the other. However, that is not sufficient. Notice that the values in column Customer Account.BID, which is a foreign key of column Broker.BID, are a *random subset of a prefix* of Trade.TID. Hence, the inclusion test adapted as above would still incorrectly report (Customer Account.BID, Trade.TID) as a foreign/primary key pair. To complicate matters further, this problem is not limited to numerical attributes. It arises with date-time fields that may contain consecutive values, or even alphanumeric fields composed of letters followed by a number (e.g., A-1, A-2). The same is true for multi-column keys. For example, Holding.(CID, SMB) is a two-column foreign key of Holding Summary.(CID, SMB). However, Broker.(BID, STID) is not a valid foreign key of Trade History.(TID, STID), even though column-wise inclusion is satisfied.

Reducing the number of false positives is a critical requirement

Proceedings of the VLDB Endowment, Vol. 3, No. 1

in order to make the identification of useful relationships feasible. As we show in the experimental section the number of false positives (i.e., pairs of columns that satisfy inclusion but are not valid fk/pk constraints) can be in the order of hundreds. Even for domain experts the task of sifting through and manually validating candidates is overwhelming. Previous work has proposed heuristic rules to reduce the number of false positives by identifying important properties that a good foreign key should satisfy. A comprehensive list of such properties, compiled based on extensive experimentation, appears in Rostin et al. [17]. Some of the most important rules are: 1. A foreign key should have significant cardinality; 2. A foreign key should have good coverage of the primary key; 3. A foreign key should not be at the same time a primary key for too many other foreign keys; 4. The set of values of a foreign key should not be a subset of too many primary keys; 5. The average length of the values in foreign/primary key columns should be similar (mostly for strings); 6. The primary key should have only a small percentage of values outside the range of the foreign key; 7. The column names of foreign/primary keys should be similar. Indeed, a lot of previous work, especially in the realm of schema matching, has used similar rules to find associations between columns [10].

It is important to note that counter-examples exist for any rule one tries to devise. One can easily come up with such examples for rules 1 and 2. A counter-example for rules 3 and 4 is when social security numbers or telephone numbers are used in a database as primary keys. Such keys are expected to appear in a large number of tables. Kang and Naughton [10] give several counter-examples for rule 7, i.e., columns with no meaningful association but very similar names.

In this work, we propose a novel method for measuring the likelihood that a pair of columns that satisfy inclusion is a useful fk/pk constraint. Our approach subsumes a variety of previous rules, and, as we show in Section 5, is both highly scalable and accurate.

Consider the set of values in a primary key P, ordered by the natural order in the underlying domain (i.e., numerical order for numeric attributes, lexicographic order for strings). We conjecture that in most cases the values in a foreign key column F form a (nearly) uniform random sample of the values in P. For example, consider columns Broker.BID and Customer Account.BID from Figure 1. The broker ids that appear in Customer Account are expected to be "sprinkled" uniformly throughout the ordered set of all broker ids. This is because we have no reason to expect a correlation between the semantics of the foreign key constraint ("this broker works with these customers") and the mechanism through which the broker ids are generated. For example, ids may be consecutive numbers between 1 and 100 generated via auto-increment. By contrast, the subset of broker ids that appear in Customer Account may reflect, say, those brokers with great reputation. It is highly unlikely that a database instance is designed such that a foreign key is a biased sample of the respective primary key (e.g., a prefix or a suffix in the ranked order). Even if this is the case, for dynamic databases the distribution of fk/pk values is expected to change over time, eventually eliminating such bias. The closer a column F is to being a uniform random sample of a primary key P, the higher we consider the likelihood that (F, P) is a useful fk/pk constraint.

Randomness is a strong requirement that implies rules 1-6: If F is a uniform random sample of P, rule 2 (and by extension rule 1 relative to the cardinality of P) is satisfied. Similarly for rule 6. If the underlying distribution of column F is the same as column P, and F is a random sample of P, then the probability that a substantial number of columns F' are random samples of F, without any real correlation between F and F', is very small (rule 3). Similarly, if F is a random sample of P, and F is a random sample of some other column P' with the same underlying distribution as P, then P and P' are clearly highly correlated. First, it is unlikely that a large number of such correlated columns P' exist (rule 4). Second, any such association (F, P') has high confidence if (F, P) has high confidence, so it is equally valid. Finally, if F is a random sample of P rule 5 is straightforwardly satisfied.

Nevertheless, one can come up with counter-examples for the randomness rule as well. Consider a data warehouse that contains a table P with all historical transactions, and a table F that references only the last month of transactions (for the purpose of efficiently answering queries on the latest data). If transaction identifiers are assigned using an auto-increment field, then the transaction id field in table F is a foreign key to table P and the transaction id values in F form a suffix of the ids in P. Note that this example also invalidates rule 6. Unless a foreign key constraint is specified in the schema, no formal method can decide *with 100% certainty* whether a column F is a foreign key with respect to primary key P. As mentioned above, useful fk/pk relationships are often data-dependent, and may not be specified in the schema. Hence, we cannot expect to find a solution with 100% precision/recall. However, as we show via extensive experiments over a large number of real databases, the randomness rule eliminates a very large number of false positives in practice.

In this paper we show that *Randomness* efficiently discovers meaningful foreign keys, including multi-column foreign-keys (which have not been considered by previous work). Our experiments show that our approach has higher accuracy than previously proposed methods, scales to very large datasets, and does not require any prior knowledge of the data (in contrast with the method in [17]). Our contributions are as follows:

- We define a distance measure between distributions, which allows us to quantify *Randomness*. This leads to a novel foreign key discovery rule that prunes a large number of false positives.
- We design fast approximate algorithms for evaluating randomness over a large set of columns, using quantile summaries.
- We design an I/O efficient algorithm for discovering single and multi-column foreign keys, which requires only two linear scans of the data. It outputs a list of fk/pk pairs, in descending order of their randomness scores. The score reflects the likelihood that the pair is a useful foreign key constraint.
- We present a comprehensive experimental validation of our approach using a large number of real and synthetic datasets.

2. PRELIMINARIES

Let $\mathbf{T}$ be a collection of relational tables, possibly from multiple databases. Let $\mathbf{C}$ be the set of all columns in $\mathbf{T}$. For the remainder of this paper, we use F (P) to refer to both a single and multi-column foreign (primary) key. Abusing notation, F and P refer both to the names of the columns (or multi-columns) and to their respective set of distinct values (or tuples). In general, F and P are multi-sets. Let $|X|$ be the number of *distinct* values in multi-set X (or distinct tuples if X is multi-column).

We assume that the single and multi-column primary keys in $\mathbf{T}$ are known, either from schema specification or from a preprocessing phase. In the latter case, Gordian [19] can be used to compute them. Let $\mathbf{P}_s$ ($\mathbf{P}_m$) denote the set of single-column (multi-column) primary keys and $\mathbf{P} = \mathbf{P}_s \cup \mathbf{P}_m$.

Similar to previous work, in order to cope with data inconsistencies, we relax the inclusion property that a foreign key must satisfy.

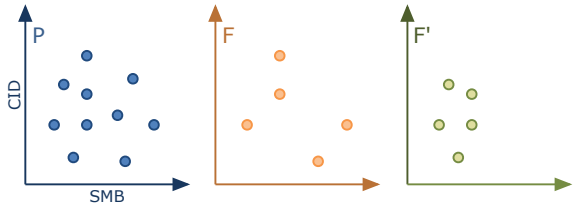


Figure 2: A good foreign key F is a set of random values from the primary key. Column F' fails the randomness test.

More precisely, we require that

$$\sigma(F, P) = \frac{|F \cap P|}{|F|} \geq \theta,$$

where $\sigma(F, P)$ is the *inclusion coefficient* and θ is user-defined. In our experiments, we use $\theta = 0.9$, i.e., partial inclusion is satisfied if at least 90% of the values in F are also contained in P. We use the notation $F \subset_\theta P$ to denote that $\sigma(F, P) \geq \theta$.

Computing $\sigma(F, P)$ is very expensive, especially when considering the potentially very large number of multi-column candidate fk/pk pairs. Therefore, we estimate all inclusion coefficients by computing a bottom-k sketch [5] for each column (refer to Appendix C for details on bottom-k sketches). We then use the SCS estimator from [5], which estimates the Jaccard coefficient $\rho(F, P) = \frac{|F \cap P|}{|F \cup P|}$. Since $\sigma(F, P) = \frac{\rho(F,P)}{\rho(F \cup P, F)}$, we estimate $\sigma(F, P)$ by dividing the estimators for the two Jaccards. [1] Section 4 provides details on how to efficiently compute bottom-k sketches for both single and multi-column candidate keys.

3. RANDOMNESS

In this section we assume that the inclusion coefficients between all pairs of single/multi-column pks and columns in $\mathbf{C}$ have been computed, and pairs that do not satisfy partial inclusion have been discarded. As mentioned in Section 1, we conjecture that randomness is a strong indicator of the quality of an fk/pk pair. Formally:

DEFINITION 1 (RANDOMNESS TEST). *Given two sets of values (tuples) F and P, test the statistical hypothesis that the distinct values (tuples) in F have the same underlying distribution as the distinct values (tuples) in P.*

Figure 2 shows an example of a two-column primary key and two candidate foreign keys. Set F is a good fk, since it appears to be a random subset of values from the pk. Set F' is a contiguous subset of the pk and does not pass the randomness test.

Domain Order. The randomness test requires the existence of an underlying order over the domain of the primary and foreign keys. To see this, consider the example in Figure 3. If the values are sorted numerically, then the candidate column F is a prefix of the primary key. However, when the same values are sorted lexicographically, F falsely appears to be a random sample of the primary key. To handle this issue, we adopt the following natural convention: numeric values are sorted numerically, and strings are sorted lexicographically. The implicit assumption is that it is very rare that a column containing only numeric values is a foreign key for a primary key that contains strings (in which case it should have been sorted lexicographically, rather than numerically). When columns contain both numeric, alphanumeric, and string values, we use a combination sort (same as the Unix "sort -n" command). For multi-column keys we define an order along each dimension, as above.

[1] In Section 5 we discuss two alternative estimators we considered. Each had a significant drawback. By contrast, this estimator proved highly accurate.

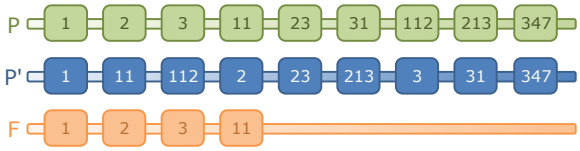


Figure 3: A column containing numeric values might falsely appear to be a random sample of a primary key based on lexicographic sorting of values.

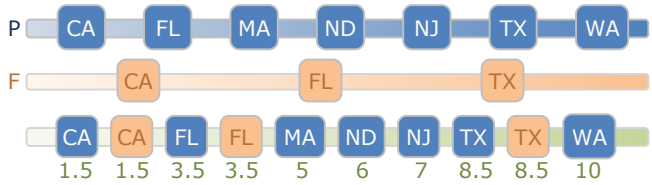


Figure 4: The Wilcoxon test: 1. Sort values in multi-set $F \cup P$; 2. Assign ranks; 3. Compute the rank-sum of values in F (13.5 in this example).

Randomness measure. A standard, non-parametric statistical test for randomness is the Wilcoxon rank-sum test [18]. Assume that F, P are single-column candidate keys. Sort the values in the multi-set union $F \cup P$ and rank them. Since $F \subset_\theta P$, the majority of values in F appear in P, so there are duplicate values. Assign the mean rank for duplicate values (i.e., if a duplicate value is 3rd and 4th in the sorted order, it is assigned rank 3.5; see Figure 4). Finally, compute the sum of ranks of all values in F. This rank-sum is an indication of whether F and P are drawn from the same distribution. Intuitively, if the rank-sum is too small, then most values in F are contained in a prefix of P, and if the rank-sum is too large then most values in F are contained in a suffix of P.

The Wilcoxon test is straightforward for univariate distributions but does not generalize to multivariate distributions, so it cannot be used for multi-column keys. Attempting to apply the Wilcoxon test separately for each dimension of a multi-column key results in false negatives. For example, consider the multi-column key F in Figure 2. Even though F appears to be a uniform random sample of P, the projection of F in either dimension is not a uniform sample due to the multiplicity of some of the values (two points project into the same value in both dimensions). An independent Wilcoxon test in either dimension would dismiss F.

We now propose a novel approach for deciding whether two multi-dimensional sets are drawn from the same distribution. Our method computes a value that reflects how close the distributions of the two sets are. We start by defining a probability distribution for each set, so that the total probability mass is 1 (this step is detailed later in the section). A standard distance measure between two probability distributions is the Earth Mover's Distance (EMD) [16]. The smaller the value $EMD(F, P)$ is, the closer the distributions of F and P are. The output of our algorithm is the list of (F, P) pairs, in increasing order of their (normalized) EMD values.

Intuitively, EMD measures the amount of work needed to convert the set of values of the foreign key into the set of values of the primary key. If we regard each distribution as piles of dirt spread over some space, EMD is the least amount of effort needed to convert the first set of piles into the second. The effort is the amount of dirt that needs to be moved times the distance it has to travel. Figure 5 illustrates the computation of EMD for pairs (F, P) and (F', P) from Figure 2. In this example, all points in a set have equal probability and the sum in each set is equal to 1. To convert F into P, a probability mass of 0.1 needs to be moved from each point $p \in F$ to the nearest point $np(p) \in P \setminus F$. Similarly for

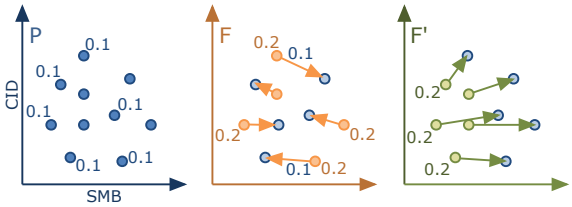


Figure 5: EMD quantifies the amount of work required to convert one set of values into another.

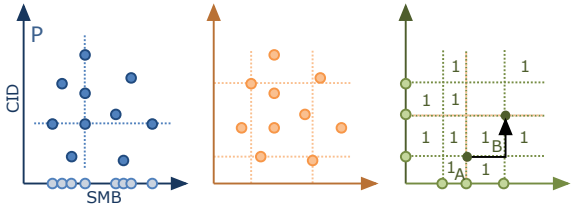


Figure 6: Constructing a 2-dimensional 4-quantile histogram for primary key P.

F' and P. Since the points of F are uniformly distributed over P the average distance between p and $np(p)$ is smaller than the average distance for F'. Hence, the amount of work needed to convert distribution F to P is smaller than the one to convert F' to P.

While the definition of EMD applies to single and multi-dimensional sets, it has a crucial restriction: unlike the Wilcoxon test, EMD requires a metric distance between the values of the two distributions. A metric distance can be used only when both columns F and P contain numeric values, but not when they contain strings. Even for numeric values, using the underlying distance is undesirable because we need to be able to compare EMD values between different candidate pairs for sorting pairs according to confidence. However, given distinct F, F', P, P', if F and P have larger ranges of values than F' and P', then $EMD(F, P)$ will generally tend to be larger than $EMD(F', P')$, even if F is a "more random" subset of P than F' is for P'. Therefore, a uniform way of defining a distance function for numeric and string columns is needed, which is independent of the range of values in any column.

We propose using the distance between the ranks of the values in the pk column. For single-column F and P, rank all values in P in the underlying ordered space, then define the rank distance between two values in F or P to be the (absolute) difference between their ranks in P. For multi-columns F and P, define the rank distance to be the sum of single-dimensional rank distances (i.e., the Manhattan distance). However, the rank distance will still introduce bias when comparing $EMD(F, P)$ and $EMD(F', P')$ if the number of values in $|P|$ is much larger than the number of values in $|P'|$. Therefore, the rank distance is normalized by the number of values, in effect replacing ranks by quantiles:

DEFINITION 2 (QUANTILE DISTANCE). *Given a multi-column set X consisting of n columns, a total order in each column X_i, a function $q_i(x)$ that returns the quantile order of value x in column X_i, and two tuples $v, w \in X$, the quantile distance is*

$$d(v, w) = \sum_{1 \le i \le n} |q_i(v) - q_i(w)|.$$

Notice that the quantile distance is independent of the type of values in X as long as a total ordering of the values *in each dimension* is defined. We refer to the EMD measure using the quantile distance as *Quantile-EMD*. A final normalization is needed to compare (F, P) and (F', P') when they have different dimensionality. Let $EMD_n(F, P) = EMD(F, P)/n$, where n is the dimensionality of F and P.

Computing Quantile-EMD. We now consider the problem of efficiently approximating $EMD(F, P)$ for all pairs of candidate keys (F, P). The first step is to define a probability distribution for F and P. The easiest choice is to let each value in F have a probability mass of $1/|F|$, and each value in P have a probability of $1/|P|$. Computing EMD is equivalent to the well-known transportation problem and can be solved by the Hungarian algorithm [7]. However, the Hungarian algorithm has cubic complexity and is very inefficient over large F and P. For our purposes, it is sufficient to compute EMD on coarser probability distributions.

More precisely, we use a quantile histogram to define the probability distribution in the primary key, since quantiles best approximate the original distribution w.r.t. the quantile distance. The probability distribution in the candidate foreign key is then defined with respect to the quantiles of the primary key.

For every single/multi-column key $P \in \mathbf{P}$ construct a *quantile histogram* based on the ℓ-quantiles of P (for some constant ℓ). In one dimension, the histogram is equi-depth. In multiple dimensions, compute quantiles separately on each dimension (over the *distinct* values in that dimension) and construct a grid based on the quantiles in each dimension. An example 2-dimensional 4-quantile histogram is shown in Figure 6. Notice that in this particular example there exists a three point tie in each dimension. After projecting the points in either dimension there are only 8 distinct values left. Hence, the 1st 4-quantile is the point with rank $8 \cdot 1/4$, the 2nd is the one with rank $8 \cdot 2/4$, etc. The probability distribution of P based on the corresponding histogram is defined as:

DEFINITION 3 (QUANTILE HISTOGRAM). *Given a multi-column primary key P consisting of n columns, let $Q_i = \{q_1^i, \ldots, q_{\ell_i}^i\}$ be the ℓ_i-quantiles of P in column i (different columns may have different number of quantiles). Let $G_P = Q_1 \times \ldots \times Q_n$ be the corresponding n-dimensional quantile grid. The quantile histogram $\bar{P}$ is defined as the number of values of P within each grid cell of G_P. The total number of grid cells is $|G_P| = \ell_1 \times \ldots \times \ell_n$. The probability distribution over P is defined as the normalized $\bar{P}$; i.e., the count in each cell is divided by $|P|$.*

For a candidate multi-column F, the probability distribution histogram based on the quantile grid G_P of P is defined as:

DEFINITION 4 (DISTRIBUTION HISTOGRAM). *Given a candidate pair (F, P), the distribution histogram $\bar{F}_P$ of F with respect to P is defined as the number of distinct values of F within each grid cell of the quantile grid G_P. The probability distribution over F is defined as the normalized $\bar{F}_P$; i.e., the count in each cell is divided by $|F|$.*

We now describe how to approximate $EMD(F, P)$ using the quantile histograms. Assume that the probability mass of a grid cell is concentrated in its upper right corner. Therefore, the distance between two grid cells is defined as the quantile distance between the upper right corners of the cells. For example, in Figure 6, the distance between grid cells A and B is $(3/4 - 2/4) + (2/4 - 1/4) = 0.5$. As before, the Hungarian algorithm is used to compute the EMD between the two distributions. The input size is now $|G_P| = \ell^n \ll |P|$ (usually $1 \le n \le 4$ and ℓ is small). Once the normalized histograms are computed, the method requires no additional access to the raw data. Note that the value ℓ need not be the same for all primary keys. Since Quantile-EMD uses the quantile distance we can compare $EMD(F, P)$ and $EMD(F', P')$ even if the quantile histograms were computed for different values of ℓ. This is important, since some primary keys may have only a few values. On the other hand, a larger ℓ for larger primary keys will improve accuracy.

Now we can bound the approximation error for the Quantile-EMD in the grid space G_P versus the initial space P (the proof is

in Appendix D).

LEMMA 1. *Let n be the space dimensionality, $1/\ell$ be the side length of the cells in G_P (in every dimension without loss of generality), and $EMD_{n,P}$, EMD_{n,G_P} be the normalized EMD in the primary space P and reduced space G_P respectively. Then*

$$|EMD_{n,P} - EMD_{n,G_P}| \leq 2/\ell.$$

4. DISCOVERING FOREIGN KEYS

Throughout this section, we use the notations from Section 2 (see also Table 5 in the Appendix). To discover foreign keys, we first compute inclusion between all pairs of primary keys and columns in $\mathbf{C}$ and then evaluate randomness only on the pairs satisfying inclusion. We accomplish this by computing bottom-k sketches and quantile histograms with two passes over the data. A pseudocode of the algorithm described below appears in Appendix B.

For single-column candidate foreign keys and single/multi-column primary keys, the bottom-k sketches can be computed in one linear scan of the database. Multi-column candidate foreign keys are challenging due to their potentially large number. However, if P is a primary key consisting of columns $(C_1, \ldots, C_n)$ and F is a candidate foreign key consisting of columns $(C'_1, \ldots, C'_n)$, then $\sigma(F, P) \leq min_{i=1}^{n} \sigma(C'_i, C_i)$. Hence, it is sufficient to consider only candidates F such that $\sigma(C'_i, C_i) \geq \theta$, for all i, where all C'_is belong to the same table. We expect only a small number of pairs (F, P) to have these properties (and confirm this experimentally). For such pairs (F, P), we compute the bottom-k sketch of F and estimate $\sigma(F, P)$, with a second pass over only the relevant columns in F (recall that the sketch of P has been computed during the first pass).

Quantile histograms for a single-column primary key P can be *computed exactly* in one linear scan if there exists an index on P – as is usually the case for pks – by reading P in sorted order and computing the quantiles incrementally (this requires knowledge of $|P|$ which can be found from table statistics). If an index does not exist we can *approximate* the quantiles in linear time using quantile summaries [9]. The distribution histograms of single-column candidate fks can be trivially computed in linear time after the quantile histograms of all pks have been computed and stored in memory; for an fk F we simultaneously compute all histograms w.r.t. all pks P for which (F, P) passes inclusion. In our experiments, the average number of such pairs, for a fixed F, was less than 10.

For a multi-column primary key P, the quantile histogram requires two passes over the data. In the first pass, we compute the quantile grid for each column $C \in P$ (either using an index or a quantile summary) and construct a multi-dimensional quantile grid. A subtle point here is that a column C of a multi-column pk might not be a pk itself; estimating quantiles on the distinct values in C requires using duplicate insensitive quantile summaries [6]. In the second pass, we scan P and populate the quantile grid. We also scan each multi-column fk F and simultaneously compute all relevant distribution histograms. In our experiments, the average number of such pairs, for a fixed F, was less than 5.

We now summarize each of the two linear scans:

Phase 1. Read all columns in table-wise order (i.e., row by row) and build bottom-k sketches for all columns in $\mathbf{C}$, as well as for all multi-column primary keys in $\mathbf{P}_m$. Also build quantile grids for all single/multi-column primary keys. All structures are stored in two hash tables B (for bottom-k sketches) and Q (for quantile/distribution histograms), using the name of the column(s) as the hash key. Evaluate all (single-column) inclusions between $F \in \mathbf{C}$ and $P \in \mathbf{P}_s$ and store candidate pairs in $\mathbf{F}_s$. Finally, evaluate (single-column) inclusions between $C \in \mathbf{C}$ and $C_i \in P, P \in \mathbf{P}_m$ and store candidates in $S(C_i)$.

Phase 2. For each multi-column pk $P = (C_1, \ldots, C_n)$, consider the n sets $S(C_i) = \{C \in \mathbf{C} \mid \sigma(C, C_i) \geq \theta, 1 \leq i \leq n\}$ computed in Phase 1. Then $F = (C'_1, \ldots, C'_n) \in S(C_1) \times \ldots \times S(C_n)$ is a candidate fk for P if there exists a table T s.t. $\forall 1 \leq i \leq n : C'_i \in T$. Compute, for each multi-column pk P the set of its candidate foreign keys and insert pairs (F, P) in $\mathbf{F}_m$. This requires access only to the sets $S(C_1), \ldots, S(C_n)$, which are stored in memory.

Next, for each $(F, P) \in \mathbf{F}_s$, scan each single-column F and compute its distribution histograms w.r.t. all relevant primary keys P. Compute $EMD_1(F, P)$ and store it in memory. For each multi-column candidate $(F, P) \in \mathbf{F}_m$ scan F and compute its multi-column bottom-k sketch as well as its distribution histograms w.r.t. all P. For each such P, verify whether (F, P) satisfies inclusion (recall that the multi-column bottom-k sketch of P was computed in Phase 1). If (F, P) does not pass the test, discard the distribution histogram $\bar{F}_P$ and remove (F, P) from $\mathbf{F}_m$. Finally, compute $EMD_n(F, P)$ for all $(F, P) \in \mathbf{F}_m$ and return $\mathbf{F} = \mathbf{F}_s \cup \mathbf{F}_m$, sorted in increasing order of EMD_n values.

5. EXPERIMENTAL EVALUATION

We evaluate our algorithm for discovering fk/pk constraints on two benchmark synthetic databases (TPC-E and TPC-H), as well as on two real datasets: a Wikipedia (WP) snapshot from March 2008 and an IMDB snapshot from January 2010. All these datasets come with a schema specification (see Appendix E for details). We implemented our algorithms in C++, and performed the experiments on an Intel Core2 Duo 2.33 GHz CPU with 4GB RAM running MySQL. We use three standard accuracy measures to evaluate our method: *precision*, *recall* and *F-measure* (the harmonic mean of precision and recall). Each measure is applied to two sets of fk/pk constraints: the "golden standard" set specified in the schema, and the top-$X\%$ constraints reported by our algorithm (for various X). We start by evaluating the accuracy of EMD computation, then evaluate the overall algorithm for both accuracy and scalability. In addition, we discuss how the results change if we also take into account the similarity of column names. Finally, we compare our results with the machine learning approach of Rostin et al. [17], which uses the 7 rules discussed in Section 1. For completeness, we also include experiments on bottom-k sketches, which show that the partial inclusion estimator is highly accurate and does not influence the overall results.

5.1 EMD Computation

By Lemma 1, the error in the computation of $EMD(F, P)$ is bounded by $2/\ell$, where ℓ is the number of quantiles in each dimension. In practice, this error is negligible even for small ℓ. Table 1 shows the average difference between the EMD computed using $\ell \in \{4, \ldots, 1024\}$ quantiles and 2048 quantiles. The averages are over all columns in the TPC-H database that pass inclusion. The reason we do not compute the differences to the actual EMD value (i.e., using all quantiles) is that EMD has cubic complexity, which is very expensive to compute when the primary key has $\Omega(10^4)$ values. Clearly, the EMD values converge very quickly as ℓ increases. Therefore, small values of ℓ are sufficient.

The last column of Table 1 shows that, when using 256 *approximate quantiles*, the difference is only 10^{-4} bigger than for 256 exact quantiles. This is not surprising, since the quantile histograms of foreign keys should remain roughly the same over small shifts in the quantile grids. The main difference between approximate versus exact quantiles is computation time. To our surprise, computing exact quantiles was about 20% faster! The reason is that the vast majority of primary keys are indexed, so using an ORDER BY

ℓ		4	16	64	256	1024	256(A)
Diff		0.06	0.009	0.002	$3\ 10^{-4}$	$4\ 10^{-5}$	$4\ 10^{-4}$
Top-25%	R	1	1	1	1	1	1
	P	0.9	0.9	0.9	0.9	0.9	0.9
Top-20%	R	0.78	0.78	0.89	0.89	0.89	0.78
	P	0.88	0.88	1	1	1	0.88
Top-15%	R	0.56	0.67	0.67	0.67	0.67	0.56
	P	0.83	1	1	1	1	0.83

Table 1: EMD accuracy for different quantile grid sizes; Diff=$EMD_{n,G_\ell} - EMD_{n,G_{2048}}$.

Dataset	TPC-H		TPC-E		WP		IMDB	
SC-FK	9		44		10		8	
MC-FK	1		1		0		0	
$\theta =$	0.9	1	0.9	1	0.9	1	0.9	1
Cand. SC	38	34	304	214	12	8	24	24
Cand. MC	1	1	4	3	0	0	0	0

Table 2: Number of candidate pairs that satisfy inclusion; SC=single-column, MC=multi-column.

SQL query had the complexity of a linear scan. By contrast, computing approximate quantiles incurred the overhead of maintaining the quantile summaries [9]. Henceforth, all reported results are for exact quantiles.

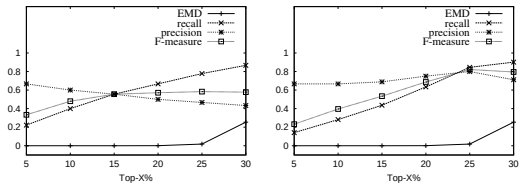
Since normalized EMD values determine the order in which we report fk/pk pairs, we also measured the effect of quantile grid sizes on the precision and recall of the final results. Smaller grid sizes, as well as approximate quantiles, do impact both precision (P) and recall (R) for the top-15% and top-20% reported pairs (see last 4 rows in Table 1). However, they have no effect for the top-25% pairs. The reason is that EMD values are very close to each other within the top-25% pairs, so even small changes impact the order of results. However, pairs below top-25% have EMD values at least one order of magnitude larger. As we discuss below, we use precisely this jump in EMD values to determine the best set of constraints to present to a user. For TPC-H, the best set is the top-25%, which remains unchanged for all grid sizes.

5.2 Overall Algorithm

For all experiments in this section, we use $\ell = 256$ quantiles for single-column primary keys, and $\ell = 16$ quantiles per dimension for multi-column primary keys. Table 2 shows the number of candidate fk/pk pairs that pass the inclusion test for each dataset. This illustrates the large number of false positives our algorithm must eliminate. For example, 217 pairs pass inclusion for TPC-E (214 single-column pairs and 3 multi-column pairs). Of these, only 45 are specified in the schema.

To evaluate the utility of our method, we measure the precision, recall and F-measure after selecting the top-$X\%$ results as the answer set, and comparing them with the golden standard specified in the schema. We report the results in two groups, for reasons that we explain later in this section.

TPC-H, WP and IMDB: Figure 7 shows the results for these three databases. A larger X (i.e., a bigger answer set) implies a larger number of false positives, hence lower precision. On the other hand, a smaller X has more false negatives (i.e., undiscovered fk/pk constraints), and thus worse recall. There is a sweet spot, which depends on each dataset, that balances precision and recall. That spot also corresponds to a big jump in the respective EMD values; we illustrate this by plotting the EMD values on the same graph. For TPC-H the sweet spot occurs at $X = 25$, for IMDB at $X = 35$, and for WP at $X = 80$. Thus, by examining the significant jumps in EMD values, the algorithm automatically proposes one or more answer sets deemed "relevant" (note that for



(a) TPC-E. (b) TPC-E extended.

Figure 8: Utility measures on TPC-E using the golden standard and extended constraints.

Foreign Key		Primary Key	
Column	Values	Column	Values
Exchange.ex_ad_id	1-4	*Address.ad_id*	1-7504
Company.co_ad_id	5-2504	*Address.ad_id*	1-7504
Customer.c_ad_id	2505-7504	*Address.ad_id*	1-7504
Trade.t_st_id	*B*	*Status type.st_id*	A,. . .,E
Broker.b_st_id	*A*	*Status type.st_id*	A,. . .,E
Company.co_st_id	*A*	*Status type.st_id*	A,. . .,E
Customer.c_st_id	*A*	*Status type.st_id*	A,. . .,E
Security.s_st_id	*A*	*Status type.st_id*	A,. . .,E

Table 3: False negatives in TPC-E (A=Active, B=Completed, C=Canceled, D=Pending, E=Submitted).

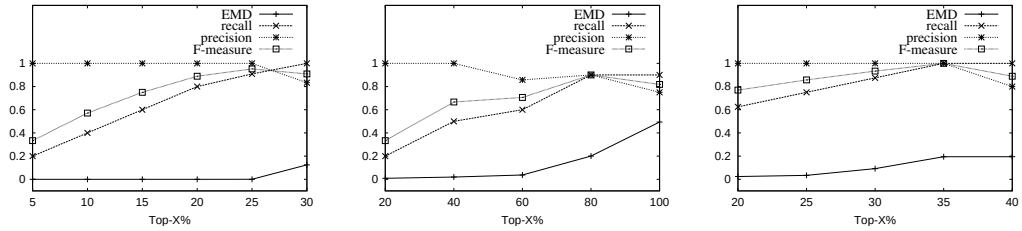
WP, an EMD jump also occurs for $X = 60$). An answer set can then be verified either experimentally (e.g., by running queries and testing if the results are meaningful), or by a domain expert.

For all three datasets in Figure 7, we achieve F-measure above 0.8 at their respective sweet spot X. For TPC-H, the answer set (top-25%) has only one false positive: the pair of columns PartSupp.PS_AVAILQTY and Supplier.S_SUPPKEY, the first containing values from 1 to 9999 and second from 1 to 10000. Clearly, only a supervised algorithm would be able to recognize this false positive. For IMDB, the answer set is exactly the golden standard.

For WP, the loss in recall originates from an unlikely source. The schema specifies that ImageLinks.il_to is a foreign key to the primary key Image.img_name. However, its inclusion coefficient is only 51%. Since we set the inclusion threshold at 90%, this results in a false negative. (We note that three other specified fk/pk pairs have inclusion coefficient below 1, but higher than 0.9). Such data inconsistencies verify the intuition that real data does not always follow the ideal rules from database theory.

TPC-E: For this database, we report two sets of experiments. In the first set (Figure 8(a)) we measure precision/recall with respect to the golden standard specified in the schema. Notice that the F-measure is below 0.6 across the board. A careful analysis of the data reveals that the reported accuracy is misleading: many false positives occur either because of *symmetry* or *transitivity*. Symmetry refers to the case when a pair of primary key columns (A, B) is specified in the schema as fk/pk in one direction, but not in the other. However, (B, A) is clearly a valid constraint in this case. Transitivity occurs when, for three columns A, B and C, with B and C being primary keys in their respective tables, the constraints (A, B) and (B, C) are specified in the schema, while (A, C) is not (although it is clearly valid). By applying symmetry and transitivity rules, we extend the set of valid constraints against which we test our results. Reporting precision/recall with respect to this augmented set of constraints improves results significantly; see Figure 8(b).

Nevertheless, none of the measures reaches 1. We attribute this to the data generation process itself. A total of 15 false positive



(a) TPC-H. (b) Wikipedia. (c) IMDB.

Figure 7: Utility measures on TPC-H, Wikipedia and IMDB.

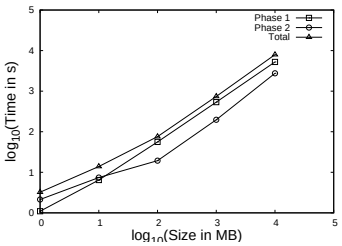


Figure 9: Scalability results.

pairs (some pairs are counted in both directions) are between only eight columns. These columns belong to seven different tables and contain exactly the same number of rows and number of distinct values: the numbers 1 to 5000. Clearly, only a domain expert can label them as false positives (in our case, we used the extensive TPC-E documentation). The algorithm also fails to discover 8 out of 45 true constaints and 28 implied constraints; the pairs are shown in Table 3. Five of these foreign keys contain only one distinct value (either the status 'Completed' or 'Active'). Clearly, the generator assigns a default value for this column for every row in the table, since not all trades in Trade can be completed, while all trades in Broker are active. One column contains address identifiers 1 to 4 even though the corresponding primary key contains 7504 distinct addresses. Finally, the other two address columns are (almost) a prefix and a suffix of the primary key and constitute a counter-example for the randomness rule.

5.3 Scalability

We tested the scalability of our method on TPC-H, for which it is easy to generate instances of progressively larger sizes. We used five instances with sizes 1MB, 10MB, 100MB, 1GB and 10GB. The running times for each of the two phases, as well as the total time are shown in Figure 9. For readability, we use a logarithmic scale on both axes. As expected, each phase takes linear time. The second phase is faster because we only have to scan the columns that satisfy inclusion (while in the first case, we scan all columns). For the 10GB instance, the total running time is less than 2.5 hours, making our method applicable to enterprise-scale datasets.

5.4 Column Names

So far our foreign key discovery process has been a data-driven approach. However, it is easy to enhance it by considering the orthogonal approach of looking at the column names (rule 7 in Section 1). As shown in [10], comparing column names is not necessarily straightforward, and can lead to false conclusions. For example, in TPC-E columns that form valid fk/pk constraints have very different names, because they contain an abbreviation of the table name as a prefix (e.g., columns Trade History.th_t_id and Trade.t_t_id are an fk/pk pair; the prefixes 'th' and 't' in the column names stand for Trade History and Trade respectively). Fortunately, TPC-E has extensive documentation that explains the naming conventions, so we can delete these prefixes and compare the remaining strings. The resulting names are identical only if the pair is a valid constraint.

	Precision	Recall	F-measure
TPC-H	1	1	1
TPC-E	0.57	0.82	0.67
TPC-E Ext.	1	0.89	0.94
IMDB	1	1	1

Table 4: Results after eliminating non-matching column names.

We are not aware of any method for automatically determining which string similarity measure to use for any given schema. In Table 4, we report our results using string identity (for TPC-E, we apply this to column names after deleting their table prefixes). The results are generated as follows: First, we compute for each database the most relevant answer set, i.e., the top-X% for the best value X. We then delete all pairs from these sets whose column names are not identical, and compute the precision/recall on the resulting answer set. For TPC-E, we also report results using the extended set of valid constraints. For WP there is no single pair with identical column names, hence we exclude it from this experiment.

5.5 Comparison With Alternatives

The algorithm of Rostin et al. [17] uses a learning phase to train four different classifiers that are then used to discover single-column keys only. Each classifier uses a training set consisting of known fk/pk pairs from four out of five different databases. The goal is to learn the relative importance of rules 1-7 stated in Section 1, then apply them to the fifth database. No classifier was consistently the best across all datasets.

We compare our results over TPC-H using the results already reported in [17]. As reported in that paper, the best classifier (J48) for the TPC-H dataset results in F-measure equal to 0.95, with the average value over all classifiers being 0.915. The success of J48 for TPC-H can be largely attributed to the use of rule 7 (matching column names), making TPC-H an easy target. Our method achieves an F-measure of 1 for TPC-H when using column names (even without column names, F-measure is 0.95).

5.6 Inclusion Estimators

We considered two alternative estimators for the inclusion coefficient $\sigma(F, P)$:

1. The estimator proposed in [3], which is unbiased. However, it is defined over sketches whose sizes are a user-defined fraction of the size of the original column. This is generally too

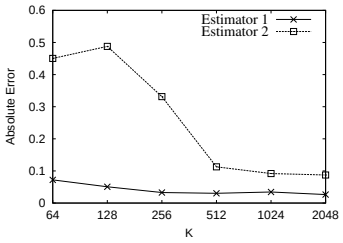


Figure 10: Accuracy of bottom-k estimators for the inclusion coefficient, as a function of k.

large for practical purposes (e.g., the Wikipedia database has size $O(10^9)$, so the size of 1%-sketches is $O(10^7)$). Note that the size of sketches impacts not just storage requirements, but more importantly, the running time for computing all inclusion coefficients.

2. Divide the estimated value of $|F \cap P|$ by the estimated value of $|F|$. Both estimated values are computed using the estimators proposed in [2]. The advantage is that these estimators work over bottom-k sketches which have constant size. Figure 10 shows an experimental comparison (over TPC-E) of this estimator, denoted Estimator2, to the one described in Section 2, which we denote as Estimator1. Clearly, Estimator1 is significantly more accurate, and it also uses bottom-k sketches. Therefore, all our experiments use Estimator1 for the inclusion coefficient. We set $k = 256$.

6. RELATED WORK

The state of the art for efficient, automatic discovery of single and multi-column primary keys is Gordian [19]. Gordian formulates the problem as a cube computation that corresponds to the computation of the entity counts of all possible column projections.

Surprisingly, very little work has dealt with discovery of foreign keys. Most work focuses on computing inclusion dependencies only [14, 4, 1, 13, 15, 12]. Recently, Bauckmann et al. [1] proposed SPIDER for efficiently detecting single-column inclusion dependencies. The algorithm first sorts the distinct values in all columns in $\mathbf{C}$ and then uses a parallel merge-sort like algorithm to compute all inclusions simultaneously. SPIDER computes inclusions exactly, but the cost is super-linear to the size of the data. A similar approach was proposed by Marchi et al. [14], using a linear pass over the data to compute an inverted index over each data type (e.g., strings, floats, integers). Subsequent passes over the index can discover single/multi-column inclusions. Marchi and Petit [15] propose a hybrid technique based on association rule mining to find low-dimensional inclusions and an optimistic exploration of high-dimensional inclusions using clique-finding. Koeller and Rundensteiner [12] utilize clique-finding for discovering high-dimensional inclusions. Partial inclusion is not addressed in these works. Lopes et al. [13] use a query workload based approach to discover foreign key relationships based on the assumption that SQL join queries use fk/pk keys. This approach is based on the availability of a query workload. Furthermore, inclusion is not a sufficient condition for foreign keys, resulting in a large number of spurious keys. Rostin et al. [17] introduced a machine learning approach for discovering foreign keys that is based not only on inclusion, but on a variety of other properties of good foreign keys. The authors use the SPIDER algorithm to discover all inclusion dependencies. Most importantly, the algorithm requires a learning step, which implies the availability of datasets with known fk/pk keys. The quality of the training dataset affects performance significantly. Finally, multi-column keys are not addressed in that work. Dasu et al. [8] proposed using minhash sketches to find potential associations between columns (or sets of columns) as a function of Jaccard coefficient. However, Jaccard is not a good indicator of inclusion coefficient when set sizes differ substantially.

7. CONCLUSION

In this work we introduced the notion of *Randomness* and showed that it can be used effectively to reduce a large number of false positive pairs produced by partial inclusion. We also provided an efficient approximation algorithm for evaluating randomness between pairs of multi-column candidate keys. We presented a combined algorithm that can discover good single/multi-column foreign keys with only two linear scans over the data. Finally, we performed a comprehensive experimental evaluation showing the efficacy of our techniques.

8. ACKNOWLEDGEMENT

The work of Meihui Zhang and Beng Chin Ooi were in part supported by Singapore NRF grant R-252-000-376-279.

9. REFERENCES

[1] J. Bauckmann, U. Leser, F. Naumann, and V. Tietz. Efficiently detecting inclusion dependencies. In *ICDE*, pages 1448–1450, 2007.
[2] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *SIGMOD*, pages 199–210, 2007.
[3] A. Broder. On the resemblance and containment of documents. In *SEQUENCES*, pages 21–30, 1997.
[4] M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. In *PODS*, pages 171–176, 1982.
[5] E. Cohen and H. Kaplan. Leveraging discarded samples for tighter estimation of multiple-set aggregates . *Joint Intl. Conf. Measurement and Modeling Comp. Sys.*, pages 251–262, 2009.
[6] J. Considine, M. Hadjieleftheriou, F. Li, J. W. Byers, and G. Kollios. Robust approximate aggregation in sensor data management systems. *TODS*, 34(1), 2009.
[7] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.
[8] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *SIGMOD*, pages 240–251, 2002.
[9] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD*, pages 58–66, 2001.
[10] J. Kang and J. F. Naughton. On schema matching with opaque column names and data values. In *SIGMOD*, pages 205–216, 2003.
[11] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *VLDB*, pages 180–191, 2004.
[12] A. Koeller and E. A. Rundensteiner. Discovery of high-dimensional inclusion dependencies. In *ICDE*, pages 683–685, 2003.
[13] S. Lopes, J.-M. Petit, and F. Toumani. Discovering interesting inclusion dependencies: application to logical database tuning. *Information Systems*, 27(1):1–19, 2002.
[14] F. D. Marchi, S. Lopes, and J.-M. Petit. Unary and n-ary inclusion dependency discovery in relational databases. *Journal of Intelligent Information Systems*, 32(1):53–73, 2009.
[15] F. D. Marchi and J.-M. Petit. Zigzag: a new algorithm for mining large inclusion dependencies in databases. *ICDM*, p.27–34, 2003.
[16] S. Peleg, M. Werman, and H. Rom. A unified approach to the change of resolution: space and gray-level. *TPAMI*, 11(7):739–742, 1989.
[17] A. Rostin, O. Albrecht, J. Bauckmann, F. Naumann, and U. Leser. A machine learning approach to foreign key discovery. *WebDB*, 2009.
[18] S. Siegel and N. Castellan. *Nonparametric statistics for the behavioral sciences*. McGraw–Hill, Inc., second edition, 1988.
[19] Y. Sismanis, P. Brown, P. J. Haas, and B. Reinwald. Gordian: Efficient and scalable discovery of composite keys. In *VLDB*, pages 691–702, 2006.

APPENDIX

A. NOTATIONS

Table 5 summarizes the notations used in the paper.

Symbol	Description
$\mathbf{T}$	Set of tables
$\mathbf{C}$	Set of columns
$\mathbf{P}_s$	Single-column primary keys
$\mathbf{P}_m$	Multi-column primary keys
$\mathbf{F}_s$	Single-column candidate foreign keys
$\mathbf{F}_m$	Multi-column candidate foreign keys
B	Hash table for bottom-k sketches
Q	Hash table for quantile/distribution histograms
F	Single/multi-column candidate foreign key
P	Single/multi-column primary key
$\widehat{C}$	Bottom-k sketch of C
$\widehat{F}$	Bottom-k sketch of F
$\widehat{P}$	Bottom-k sketch of P
$\bar{P}$	Quantile histogram of P
$\bar{F}_P$	Distribution histogram of F with respect to P
$\sigma(F,P)$	Inclusion coefficient
θ	User-defined threshold for inclusion coefficient

Table 5: Notation used throughout the paper.

B. ALGORITHM

Algorithm B.1: DISCOVER FOREIGN KEYS($\mathbf{C}, \mathbf{P}_s, \mathbf{P}_m, \theta$)

Phase 1.
$\mathbf{F}_s \leftarrow \emptyset, \mathbf{F}_m \leftarrow \emptyset, B \leftarrow \emptyset, Q \leftarrow \emptyset, S \leftarrow \emptyset$
for all $C \in \mathbf{C}:\ B[C] \leftarrow \widehat{C}$
for all $P = \{C_1, \ldots, C_n\} \in (\mathbf{P}_s \cup \mathbf{P}_m)$
  for $p \leftarrow 1$ **to** n
    do **for all** $C_f \in \mathbf{C}$
      do **if** $\sigma(\widehat{C}_f, \widehat{C}_p) \geq \theta$ /*$\widehat{C}_f, \widehat{C}_p \in B$*/
        then **if** $n = 1:\ \mathbf{F}_s \leftarrow (C_p, C_f)$
          if $n > 1:\ S[P, C_p] \leftarrow C_f$
  if $n > 1:\ B[P] \leftarrow \widehat{P}$
  /*For $n = 1$, $\widehat{P}$ already in B*/
  $Q[P] \leftarrow \bar{P}$

Phase 2.
for all $P = \{C_1, \ldots, C_n\} \in \mathbf{P}_m$
  for all $T \in \mathbf{T}$
    $\mathbf{F}_m \leftarrow (\{\{C'_1, \ldots, C'_n\} \mid C'_i \in S[P, C_i] \cap T\}, P)$
  for all $F = (\{C'_1, \ldots, C'_n\}, P) \in \mathbf{F}_m$
    do Build $\widehat{F}$
      if $\sigma(\widehat{F}, \widehat{P}) \geq \theta$ /*$\widehat{P} \in B$*/
        then $Q[P] \leftarrow \bar{P}$
          $Q[F] \leftarrow \bar{F}_P$
        else Remove (F, P) from $\mathbf{F}_m$
for all $(F, P) \in \mathbf{F}_s:\ Q[F] \leftarrow \bar{F}_P$

for all $(F, P) \in (\mathbf{F}_s \cup \mathbf{F}_m)$
  Compute $EMD_n(F, P)$ /*Using Q*/
Output $\mathbf{F} = \mathbf{F}_s \cup \mathbf{F}_m$ in increasing order of EMD_n

C. BOTTOM-K SKETCHES

Given a set F, a bottom-k sketch $\widehat{F}$ for F is computed as follows: Assign ranks to all values in F uniformly at random, and let $\widehat{F}$ be the set of k values with the smallest ranks. In practice, to compute the rank assignment we choose a hash function h, hash each value in F, and keep the k values corresponding to the smallest k hash values. If F is a set of tuples, rather than simple values, we first concatenate all values in a tuple using a predefined field separator and hash the resulting string as a whole. Figure 11 shows an example bottom-2 sketch for a set of tuples. Clearly, a bottom-k sketch can be computed in one pass over F.

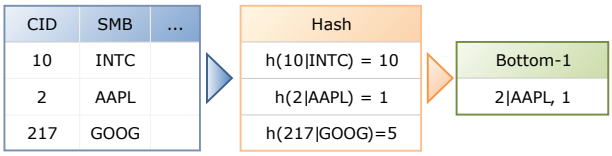


Figure 11: Constructing a Bottom-k sketch.

Bottom-k sketches have been used to estimate various measures, such as the Jaccard coefficient $\rho(F,P) = \frac{|F \cap P|}{|F \cup P|}$ (see [5]) or the intersection size $|F \cap P|$ (see [2]). The estimators require that the same hash function h be used for computing both bottom-k sketches $\widehat{F}$ and $\widehat{P}$ (hence, the sketches are called *coordinated*).

D. PROOF OF LEMMA 1

PROOF. Let $\text{EMD}_P = n \cdot \text{EMD}_{n,P}$ and $\text{EMD}_{G_P} = n \cdot \text{EMD}_{n,G_P}$ be the unnormalized EMD values in the primary space P and the reduced space G_P respectively. Consider a movement of mass m that EMD_P executes in the primary space P, from a point p to a point q. Its cost is $m \cdot d(p,q)$. Let a and b be the upper right corners of the cells that contain p, respectively q. Then we can define a valid movement of mass m in the space G_P, between a and b. The cost of this mass movement is $m \cdot d(a,b) \leq m(d(a,p)+d(p,q)+d(q,b)) \leq md(p,q) + m\frac{2n}{\ell}$. Making this transformation for all mass movements in EMD_P, we obtain a valid mass movement in G_P, of cost at most $\text{EMD}_P + 2\frac{n}{\ell}\sum m \leq \text{EMD}_P + 2\frac{n}{\ell}$ (the sum is over all the mass moved in EMD_P). Since EMD_{G_P} is the *minimum* cost movement in G_P, we deduce $\text{EMD}_{G_P} \leq \text{EMD}_P + 2n/\ell$. A similar argument holds for the other inequality, by transforming mass movements from EMD_{G_P} into valid mass movements in P. We deduce that $|\text{EMD}_P - \text{EMD}_{G_P}| \leq \frac{2n}{\ell}$. □

E. DATASET DESCRIPTIONS

The datasets can be downloaded from the following sites: TPC-H from `http://www.tpc.org/tpch`, TPC-E from `http://www.tpc.org/tpce`, WP from `http://www.archive.org/details/enwiki-20080312`, IMDB from `http://www.imdb.com/interfaces`.

When generating instances for the synthetic datasets, we use the following parameter settings: For TPC-H we use scale factor 1. For TPC-E we use 1000 customers, 20 trading days, and scale factor 1000. The characteristics of all datasets are given in Table 6, where $|\mathbf{T}|$ is the number of non-empty tables, $|\overline{\mathbf{C}_T}|$ and $\max|\mathbf{C}_T|$ are the average and maximum number of columns per table, and $|\overline{\mathbf{R}_T}|$ and $\max|\mathbf{R}_T|$ are the average and maximum number of rows per table.

All these datasets come with a schema specification. Table 7 summarizes the single/multi-column foreign/primary keys explicitly stated in each schema. Notice that, e.g., TPC-E specifies nine 2-column primary keys but only one 2-column foreign key.

F. SCHEMA AND DATA UPDATES

Our methods can easily handle insertions and deletions of new tables and columns given the existing bottom-k sketches and quantile/distribution histograms. Let the new set of columns be $\mathbf{C}'$.

	$\lvert\mathbf{T}\rvert$	$\lvert\overline{\mathbf{C}_T}\rvert$	$\max\lvert\mathbf{C}_T\rvert$	$\lvert\overline{\mathbf{R}_T}\rvert$	$\max\lvert\mathbf{R}_T\rvert$
TPC-H	8	8	16	1082504	6000003
TPC-E	32	6	24	171127	4469625
WP	15	6	16	24356005	227867141
IMDB	9	2	2	1136607	5107802

Table 6: Datasets characteristics.

		TPC-H		TPC-E		WP		IMDB	
		PK	FK	PK	FK	PK	FK	PK	FK
SC		6	9	20	44	5	10	5	8
MC	2	2	1	9	1	7	–	4	–
	3	–	–	2	–	3	–	–	–
	4	–	–	1	–	–	–	–	–
Total		8	10	32	45	15	10	9	8

Table 7: Foreign/primary keys according to schema specifications.

First, identify new primary keys and insert them in $\mathbf{P}_s, \mathbf{P}_m$. Then, re-run Algorithm B.1 on $\mathbf{C}', \mathbf{P}_s, \mathbf{P}_m, \theta$, building only the new bottom-k and quantile/distribution histograms, as necessary.

Handling data insertions and deletions on existing columns is a little harder. Existing bottom-k sketches can easily be updated under insertions only. The new values are simply hashed and inserted in the corresponding sketches if necessary. However, deletions are not straightforward: if a deleted value was part of the bottom-k sketch, a rescan of the corresponding column is needed in order to identify the new k-th minimum hash value. One way to handle deletions without rescanning the data is to maintain larger bottom-k sketches (e.g., twice as large as needed). That way, we only rescan the data infrequently. Since in practice we expect a balanced insertions and deletions workload, this simple strategy is likely to obviate the need of a rescan in most settings.

Updating the quantile/distribution histograms is generally hard, both under insertions and deletions. A small number of insertions or deletions can be accommodated by identifying the histogram cells that contain the respective tuples, and incrementing or decrementing their counters. However, if a large amount of data is inserted or deleted, the distribution of the underlying columns is likely to change. As a result, the quantile grids on each dimension also change. This requires rescaning the data in order to compute a new quantile grid and a new histogram. A simple way of reducing the cost of updates is to use the existing quantile grid for a batch of updates and rebuild it only after a certain number of updates. Depending on the application and the underlying data we can also use well known techniques to detect a change in the distribution and trigger a rebuild [11].